Universidade de Aveiro
Departamento de
Eletrónica, Telecomunicações e Informática
2012

David Miguel
Vicente Ferreira

Enriquecimento do EPG usando Semântica

"The Semantic Web is not a separate Web but an extension of the current one, that any piece of information is given well-defined meaning, better enabling computers and people to work in cooperation"

— Tim Berners-Lee

David Miguel Vicente Ferreira

Enriquecimento do EPG usando Semântica

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Cláudio Teixeira, Equiparado a Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e Professor Doutor Joaquim Sousa Pinto, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Professor Doutor José Luís Guimarães Oliveira
Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais

Professora Doutora Ana Alice Baptista
Professora Auxiliar do Departamento de Sistema de Informação da Universidade do Minho

Doutor Cláudio Teixeira
Equiparado a Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Joaquim Sousa Pinto
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Ao meu orientador, Professor Cláudio Teixeira, o meu sincero agradecimento por todo o seu apoio, conselhos e orientação ao longo deste árduo trabalho. Também ao Professor Joaquim Sousa Pinto, por toda a sua ajuda e conselhos disponibilizados.

Para a engenheira Telma Mota e para o engenheiro Ricardo Pereira o meu obrigado por toda a sua ajuda, amizade e conselhos nos problemas que foram surgindo.

Ao Miguel Grade, engenheiro de software da empresa Maisis, o meu agradecimento pelo esforço de transmitir alguma da sua experiencia, dando-me concelhos, esclarecendo-me todas as questões e problemas, sobretudo num nível mais técnico.

Um agradecimento especial a toda a minha família. Para o meu pai (José) e para a minha mãe (Maria), obrigado por todo o esforço (que foi bastante) e por todo o apoio que me deram neste meu percurso de vida.

Filipa, obrigado pela ajuda, paciência, força e carinho. Mas sobretudo obrigado por estares sempre ao meu lado.

Por fim, Tiago Nunes, Milton Silva, David Campos, João Pires e Raquel Soares, obrigado por estarem presentes na minha vida académica, partilhando boas experiências como também más. Parte do que sou hoje é graças a vocês.

palavras-chave Web Semântica, RDF, TDB, SPARQL, tripletos, DBpedia.

resumo Web Semântica é um conceito que permite chegar ao conhecimento ao invés da informação através da rede.

Através da Web Semântica é possível de uma forma mais intuitiva agrupar conteúdos. No cenário televisivo, é possível agrupar todos os conteúdos, criando métodos (ou formas) de recomendação para o telespectador. Embora algumas ferramentas semânticas ainda não se encontrem suficientemente maduras, dificultando a confiança nestes serviços, este trabalho tentou utilizar os conceitos da web semântica para fazer recomendações em serviços IPTV com o MEO.

keyword Semantic Web, RDF, TDB, SPARQL, triple, DBpedia.

abstract Semantic Web is a concept that lets you get to knowledge instead of information through the network.

Through the Semantic Web is possible to aggregate all contens in a more intuitive way. On television scenario, is possible group all contents, creating methods (or forms) of recommendation to the viewer. Although some semantic tools are not yet mature enough, making it difficult to trust on these services, this study attempted to use the concepts of the semantic web to make recommendations on IPTV services with MEO.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de acrónimos

API *Application Programming Interface*

DSP *Dynamic Semantic Publishing*

EPG *Electronic Programming Guide*

FOAF *Friend Of A Friend*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

IMDB *Internet Movie Database*

ISO *International Organization for Standardization*

JAX-RS *Java API for RESTful Web Services*

JSON *JavaScript Object Nation*

LOD *Linking Open Data*

NLP *Natural language processing*

ORM *Object RDF Mapper*

OASIS *Advancing Open Standards for the Information Society*

OSP *Object - Subject - Predicate*

OWL *Web Ontology Language*

POS *Predicate - Object - Subject*

RDF *Resource Description Framework*

ReST *Representational State Transfer*

SAPO *Serviço de Apontadores Portugueses Online*

SPARQL ***S****imple* ***P****rotocol* ***a****nd* ***R****DF* ***Q****uery* ***L****anguage*

SPO *Subject - Predicate - Object*

TDB *Triple Data Base*

URI *Uniform Resource Identifier*

URL *Universal Resource Locators*

XHTML *eXtensible HyperText Markup Language*

XML *eXtensible Markup Language*

Capítulo 1

Introdução

Atualmente, o ser humano cada vez mais se encontra dependente da Internet. Quer seja por motivos profissionais ou apenas lúdicos, vão surgindo os mais diversos dispositivos eletrónicos que permitem ao homem conectar-se à Web e nela introduzir ou adquirir diversos dados. Deste modo, com facilidade se mistura informação útil e verdadeira com informação fútil e falsa.

Sendo assim, torna-se necessário que a Web atual evolua e se apresente mais inteligente. Com a Web Semântica é possível a interligação de conteúdos, conseguindo atribuir-lhes um significado para que sejam percetíveis tanto pelo ser humano como pelo computador. Assim, permite-se que um computador seja capaz de visualizar o *mundo* tal com o ser humano, conseguindo, deste modo, verificar e validar a veracidade da informação.

1.1 Motivação

A Web atual apresenta informação usando linguagem natural, gráficos, multimédia e layouts, desta forma o ser humano consegue processar a informação encontrada deduzindo factos através de informações parciais e criando associações mentais. A Web atual dificulta a combinação de dados, pois o computador não é auto suficiente para poder utilizar informações parciais, interpretar imagens, perceber *labels* ou combinar diferentes hierarquias em *eXtensible Markup Language* (XML). Por exemplo, uma agência de viagens necessita de pesquisar em vários sites de hotéis para obter todos os dados que necessita. Devido à grande variedade de informação encontrada na Web, uma única pesquisa não garante que o resultado encontrado seja fidedigno. É sempre necessário pesquisar em variados fornecedores de informação para se garantir que esta seja verdadeira e correta. Desta forma, a introdução de tecnologias semânticas tornará a Web mais veloz, sofisticada e usável.

O seguinte exemplo reflete a importância da semântica na Web. Ambas as frases estão na forma "sujeito-verbo-objecto" uma das possíveis formas gramaticais:

1. David gosta de cães.

2. Cães assustam a Filipa.

Ambas as frases representam fragmentos de informação. Através destas duas frases consegue-se retirar que "David"e "Filipa"são pessoas, que "cão"é um animal e as palavras "gosta" e "assustam" transmitem a relação entre a pessoa e o animal. Uma vez compreendido

o sentido do verbo gostar e assustar, bem como o significado da palavra cão, as duas frases anteriores tornam-se claras, podendo desta forma, inferir novas ideias e conceitos. Graças à junção de todo este conhecimento adquire-se uma perceção e uma compreensão do mundo.

Com a futura Web, a Web 3[1], será fácil perceber qual o significado de todas as palavras porque todas as palavras serão objetos com informação associada. O primeiro passo para esta ideia se tornar realidade consiste em começar por criar pequenas fontes de informação com conteúdo semântico. Na verdade o crescimento da Web 3 será idêntico ao crescimento da Internet. No inicio irão existir pequenas fontes de dados mais centralizadas num determinado tipo de informação, para posteriormente se tornar global como a Internet o é nos dias de hoje.

O potencial da semântica encontrar-se-á nas tecnologias usadas no dia a dia. Por exemplo, a televisão atualmente fornece informação sobre os programas através do *Electronic Programming Guide* (EPG) do *Serviço de Apontadores Portugueses Online* (SAPO) (no caso da Meo). Se os dados do EPG fossem provenientes de uma base de dados semântica encontrar-se-ia mais conteúdo. Por exemplo, com a semântica facilmente se encontram os filmes em que um determinado ator participa, isto porque, os filmes bem com o ator são vistos como objetos que se interligam entre si. Por sua vez, esse ator encontra-se ligado a outros filmes em que participou. Deste modo, será possível procurar todos os filmes no EPG onde esse ator participou. Nesta situação, o *zapping* poderia ser feito de um modo bastante diferente dos dias de hoje, como também seria possível recomendar o conteúdo mais acertado para um determinado perfil de um utilizador. Para se obter este perfil caracterizador do utilizador será necessário ter em consideração os formatos e géneros preferidos, a duração habitual dos conteúdos que o utilizador costuma assistir, bem como os realizadores e os atores recorrentes nos diversos conteúdos. Desta forma, será possível a modelação do perfil do utilizador e criar um motor de recomendação de conteúdo.

Uma base de dados semântica é o ideal para um sistema deste cariz porque para além de conter toda a informação correspondente aos objetos armazenados de uma forma bem estruturada e detalhada, também, os objetos se encontram ligados entre si. A ligação entre objetos contém uma propriedade associada que elucida qual a relação entre eles. Com esta rede facilmente se encontram conteúdos semelhantes, tornando desta forma mais fácil encontrar futuras recomendações.

Este projeto encontra-se enquadrado com vários projetos pertencentes a empresas como a Maisis, Portugal Telecom e com alguns projetos de alunos da FEUP. No global constituirá uma nova funcionalidade para o interface da Meo, procurando desenvolver através de tecnologia semântica um ambiente mais agradável para um telespectador.

1.2 Objetivos

Este projeto tem como objetivo o estudo, conceção e teste de uma ferramenta que permita a integração de dados semânticos no EPG do SAPO. O objetivo geral do projeto consiste em realizar uma ferramenta que torne os conteúdos televisivos mais pessoais. Através do conhecimento dos gostos, preferências e hábitos do utilizador será possível filtrar e selecionar autenticamente os conteúdos televisivos. Com este sistema de recomendação, o utilizador terá uma experiência melhorada facilitando o processo de seleção de conteúdo. Para existir uma recomendação inteligente é necessário o conhecimento de todo o conteúdo existente. Deste modo, pretende-se criar uma *triplestore* (base de dados em formato de tripletos) que contenha o conhecimento cinematográfico existente nos canais televisivos.

Todos estes conhecimentos para além de conterem toda a informação correspondente são classificados com uma taxonomia pré-definida, proporcionando uma melhor interligação com projetos externos a este. Desta forma, os sistemas que utilizam a mesma base de dados conseguirão facilmente perceber o significado do conteúdo encontrado.

1.3 Metodologia

Este projeto foi sofrendo alterações ao longo do tempo, no entanto, nunca perdeu o mote inicial: a utilização de informação semântica. Numa fase inicial do trabalho, o objetivo passava não pela agregação de conteúdos televisivos, mas sim noticiosos. Esta informação seria agregada a partir de fontes consideradas fidedignas no *Twitter*. Estas fontes contêm *tweets* num formato idêntico, podendo assim, ser analisados, obtendo o conteúdo semântico associado.

No entanto, rapidamente se constatou a inviabilidade da solução. Além disso, os requisitos do projeto em que esta dissertação se inseriu também foram alterados, sendo então dado ênfase aos conteúdos cinematográficos.

Assim, para cumprir os objetivos do trabalho, serão abordadas experiências de utilização de dados *Resource Description Framework* (RDF) e de junção de informação de várias fontes.

1.4 Estrutura

Este documento encontra-se dividido em cinco capítulos. No capítulo 2 consta o estudo do contexto em que o projeto se insere, analisando algumas tecnologias já existentes. É também realizado um exame cuidado das soluções tecnológicas existentes, encontrando a arquitetura que, no ponto de vista do autor, melhor se adapta aos requisitos encontrados. O mesmo é feito para as linguagens de programação mais adequadas para o desenvolvimento do projeto.

No capítulo 3 é apresentada a arquitetura proposta para o sistema e o seu funcionamento geral. São ainda justificadas algumas das opções tomadas.

No capítulo 4 é abordado o funcionamento do sistema de um ponto de vista mais técnico. São, também, mencionados alguns problemas que foram surgindo na elaboração do projeto.

Finalmente, no capítulo 5 é efetuado um balanço geral sobre os conceitos abordados e sobre a solução implementada. Para além disso, são sugeridas algumas ideias de trabalho futuro.

Capítulo 2

Estado da arte

2.1 A Web Semântica

Com a Web Semântica será possível através da estruturação e conjuntos de regras de inferência obter informações deduzidas de uma forma automática. Contudo, ao contrário da Web atual, o conteúdo da Web Semântica não será processado apenas por humanos, mas também por máquinas[1]. A arquitetura da Web Semântica é definida como ilustrado pela figura 2.1 [1]. Nesta arquitetura verifica-se que a camada superior estende as funcionalidades das camadas inferiores.

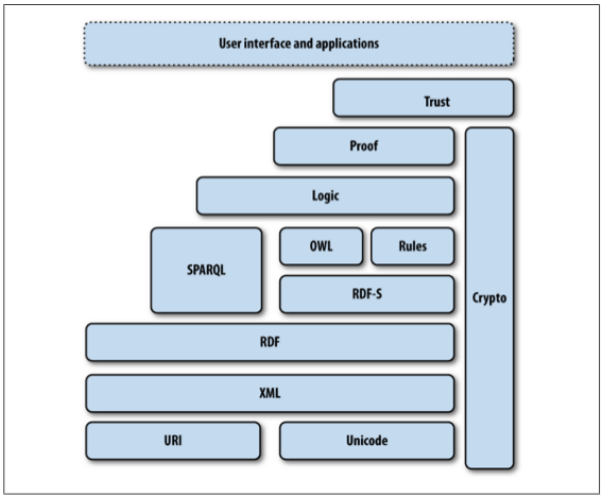

Figura 2.1: W3C semantic web technology stack[1]

A Web Semântica permitirá uma evolução da Web atual para a futura Web 3[1]. Na Web 3 procura-se que as informações estejam de tal forma organizadas, que permitam um processamento mais inteligente. Esta informação é encontrada de uma forma mais rápida e eficaz, facilitando a comunicação entre dispositivos heterogéneos.

[1]The Semantic Web of Data Tim Berners-Lee: `http://www.youtube.com/watch?v=HeUrEh-nqtU`

2.2 Resource Description Framework (RDF)

RDF é uma linguagem *standard* para expressar modelos de dados usando declarações expressas em tripletos. Deste modo, é possível a partilha desses dados entre pessoas e máquinas. RDF é recomendado pela W3C[2], o que leva a que haja uma vasta coleção de ferramentas e serviços que o utilizam(algumas nomeadas posteriormente). Através do RDF são criadas regras que originam modelos mais precisos e robustos reduzindo assim a ambiguidade. Uma grande razão para essa ambiguidade ser reduzida é o facto de RDF usar *Uniform Resource Identifier* (URI). URIs são chaves únicas que identificam um determinado recurso. Imaginando um recurso como um nó inserido num grafo, é necessário que esse nó contenha uma chave única que o identifique de modo a criarem-se tripletos com relações consistentes.

A construção de um URI consiste na definição do método (tal como "http", "ftp", "mailto", "crid"ou "file"), seguido de dois pontos e informação específica a ser processada por cada esquema. No exemplo 2.1 encontram-se alguns exemplos de URIs absolutas.

- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- ftp://example.org/resource.txt
- file:////home/example/example.org/resource.txt
- urn:issn:1535-3613

Exemplo 2.1: Exemplos de URIs Absolutas

É importante observar que o URIs não são URLs, embora cada URL seja um URI. Na prática, isso significa que não se deve assumir que quando se acede a um URI através da Internet se obtenha informação, apesar de ser boa prática a sua utilização.

Normalmente para criar um URI é indicado o próprio nome do recurso. Por exemplo, no DBpedia os URIs respetivos a filmes, é abrangido o nome do próprio filme. Mas por vezes, verifica-se a existência de vários com o mesmo nome, sendo que, o URI correspondente a um filme é maioritariamente identificado pelo nome do mesmo. Existe uma convenção para evitar a ambiguidade de diferentes recursos com o mesmo nome [1, 2, 3]. Por exemplo, o URI http://dbpedia.org/resource/Titanic_(1997_film) pertence ao filme Titanic, realizado no ano 1997. No DBpedia encontram-se mais dois *Titanic's*, um realizado em 1953 e outro que é uma série. Ambos os URIs contêm o nome Titanic mas para serem diferenciados, num foi introduzido o ano (http//dbpedia.org/resource/Titanic_(1953_film)) e no outro foi escrito explicitamente que é uma mini série (http//dbpedia.org/resource/Titanic_(TV_ miniseries)).

Os dados para poderem ter algum conteúdo semântico têm de ser guardados na forma de tripleto, isto é, na forma de recurso-predicado-objecto, em que todos eles são identificados por um URI correspondente. O tripleto é o formato mais usual, embora existam outros recursos como, por exemplo, o quadruple, sujeito-predicado-objeto-contexto. Os tripletos são armazenados numa triplestore que é semelhante a uma base de dados relacional. O armazenamento e a recuperação de dados é feito através de linguagens *query*. Para a troca entre aplicações destes tripletos é normalmente usado o formato RDF.

Os URIs podem referenciar qualquer "coisa" quer seja um recurso ou um objeto; estes URIs são conhecidos por URIref. Através de um identificador de fragmento opcional é possível identificar:

[2]http://www.w3.org

- indivíduos, e.g., José Saramago, identificado por *http://www.w3.org/People/JS/contact #me*
- tipos de objetos, e.g., Person, identificado por *http://www.w3.org/2000/10/swap/pim/ contact#Person*
- propriedades de objetos, e.g., mailbox, identificado por *http://www.w3.org/2000/10/ swap/pim/contact#mailbox*
- valores das propriedades, e.g. mailto:em@w3.org, identificado por *http://www.example.org/staffem/85741*

Um documento RDF pode também incluir **literais**. Estes literais traduzem-se em conteúdos literários, não representam um objeto, mas sim nomes, datas ou outra informação sobre um recurso em questão. Em RDF todos os literais podem ter o tipo ou a linguagem associada. O tipo é representado por um URI, por exemplo, no tipo inteiro é *http://www.w3.org/2001/ XMLSchema#int* ou *xsd:int*, para o caso da linguagem é usado *International Organization for Standardization* (ISO) 630 para especificar a língua, por exemplo *pt* para Português, *en* para Inglês e *ja* para Japonês.

Para além de literais e de URIs também existem ***blank nodes***. Os *blank nodes* são usados quando não há um identificador único do recurso em questão. Sendo assim, o *blank node* pode servir de agregador de identificadores. No caso de existir uma grande quantidade de nós associada a um recurso, e se for possível deduzir que certos nós se correlacionam de modo a poder agrupá-los, pode ser criado um novo nó, um *blank node*. Assim, os nós agrupados ficam ligados a um *blank node* que por sua vez se liga ao recurso. Através dos *blank nodes* consegue-se obter um grafo mais legível[1, 3].

RDF usa vários formatos de serialização, existindo inúmeras bibliotecas que já lidam com essa complexidade, por exemplo Jena da Apache (descrito na secção 2.8.5). Estes formatos são essencialmente quatro: N-Triples, N3, RDF/XML e RDFa.

2.2.1 N-Triples

Este formato tem a particularidade de ser o formato mais simples em anotações. Apesar da sua simplicidade, é detalhado; cada linha neste formato representa uma única instrução, contendo apenas o sujeito, predicado e o objeto, finalizando com um ponto. O sujeito e o objeto podem ser representados como anónimos, para isso, utiliza-se um *underscore*, dois pontos e um nome alfanumérico. Por exemplo *_:Nome* em que "Nome"é o nome atribuído ao objeto/sujeito (exemplo 2.2).

```
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/homepage>
   <http://kiwitobes.com/>.
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/nick> "kiwitobes
    ".
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/name> "Toby
    Segaran".
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/mbox>
   <mailto:toby@segaran.com>.
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/interest>
   <http://semprog.com>.
```

```
<http://kiwitobes.com/toby.rdf#ts> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
    type>
   <http://xmlns.com/foaf/0.1/Person>.

<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/knows> _:jamie .
<http://kiwitobes.com/toby.rdf#ts> <http://xmlns.com/foaf/0.1/knows>
   <http://semprog.com/people/colin>.

_:jamie <http://xmlns.com/foaf/0.1/name> "Jamie Taylor".
_:jamie <http://xmlns.com/foaf/0.1/mbox> <mailto:jamie@semprog.com>.
 _:jamie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://xmlns.com/foaf/0.1/Person>.

<http://semprog.com/people/colin> <http://xmlns.com/foaf/0.1/name> "Colin Evans
    ".
<http://semprog.com/people/colin> <http://xmlns.com/foaf/0.1/mbox>
   <mailto:colin@semprog.com>.
<http://semprog.com/people/colin> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
    type>
   <http://xmlns.com/foaf/0.1/Person>.
<http://semprog.com> <http://www.w3.org/2000/01/rdf-schema#label>
   "Semantic Programming".
<http://semprog.com> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://xmlns.com/foaf/0.1/Document>.
```

Exemplo 2.2: Formato N-Triples

2.2.2 N3

Os N-triples são concetualmente muito simples, mas apresentam uma grande repetição. Deste modo, as informações redundantes levam mais tempo para a transmissão e análise. O formato N3 condensa muita da repetição que surge no formato N-Triple, pela adição de algumas estruturas. Num grafo RDF cada conexão entre nós representa um tripleto. Uma vez que um nó pode participar em inúmeras relações, N3 reduz o número de repetições reconhecendo os URIs utilizados com frequência. Da mesma maneira que XML fornece um mecanismo de espaços para nomes, tornando-se nomes qualificados (qnames); N3 permite definir um prefixo identificando uma entidade um URI relativo a um conjunto de prefixos declarados no início do documento. Como por exemplo:

```
@PREFIX dbp:   http://dbpedia.org/resource/
```

Exemplo 2.3: Declaração de um prefixo.

Com esta instrução no início do documento sempre que for necessário o uso do URI *http://dbpedia.org/resource/Name*, por exemplo, basta agora usar *dbp:Name*.

Este formato também introduz o ponto e vírgula como substituto do sujeito anterior. Desta forma é reduzido muita da repetição como é encontrado no formato N-triple. Um exemplo da substituição do sujeito é demonstrado no exemplo seguinte:

```
dbp:Titanic_%25281997_film%2529 rdf:type http://dbpedia.org/ontology/Film ;
   foaf:name "Titanic"@en .
```

Exemplo 2.4: Desta forma não foi necessário repetir o sujeito.

Os *blank nodes* também são representados de uma forma melhorada. N3 não necessita de um nome conhecido para os *blank nodes*, apenas é necessário usar parêntesis retos. Desta forma, tudo o que se encontra dentro destes parêntesis corresponde aos *blank nodes* (exemplo 2.5).

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix semperp: <http://semprog.com/people/>.
@prefix tobes: <http://kiwitobes.com/toby.rdf#>.

tobes:ts a foaf:Person;
    foaf:homepage <http://kiwitobes.com/>;
    foaf:interest <http://semprog.com>;
    foaf:knows semperp:colin,
        [ a foaf:Person;
            foaf:mbox <mailto:jamie@semprog.com>;
            foaf:name "Jamie Taylor"];
    foaf:mbox <mailto:toby@segaran.com>;
    foaf:name "Toby Segaran";
    foaf:nick "kiwitobes".

<http://semprog.com> a foaf:Document;
    rdfs:label "Semantic Programming".

semperp:colin a foaf:Person;
    foaf:mbox <mailto:colin@semprog.com>;
    foaf:name "Colin Evans".
```

Exemplo 2.5: Formato N3

2.2.3 RDF/XML

Este formato foi a primeira recomendação do W3C para modelos RDF. Por conseguinte, o formato RDF/XML é vulgarmente conhecido apenas por RDF. Este formato é de leitura complexa porque inicialmente foi criado para pequenas descrições e não para grafos extensos (como hoje em dia existem)[2, 3].

RDF/XML mantém todas as regras do formato XML, todos os *namespaces* são definidos no início do documento. (exemplo 2.6)

```
<rdf:RDF
    xmlns:foaf='http://xmlns.com/foaf/0.1/'
    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
    <foaf:Person rdf:about="http://kiwitobes.com/toby.rdf#ts">
    <foaf:name>Toby Segaran</foaf:name>
        <foaf:homepage rdf:resource="http://kiwitobes.com/"/>
        <foaf:nick>kiwitobes</foaf:nick>
        <foaf:mbox rdf:resource="mailto:toby@segaran.com"/>

    <foaf:interest>
        <foaf:Document rdf:about="http://semprog.com">íprincpio
            <rdfs:label>Semantic Programming</rdfs:label>
        </foaf:Document>
    </foaf:interest>
    <foaf:knows>
```

```
        <foaf:Person rdf:about="http://semprog.com/people/colin">
            <foaf:name>Colin Evans</foaf:name>
            <foaf:mbox rdf:resource="mailto:colin@semprog.com"/>
        </foaf:Person>
    </foaf:knows>

    <foaf:knows>
        <foaf:Person>
            <foaf:name>Jamie Taylor</foaf:name>
            <foaf:mbox rdf:resource="mailto:jamie@semprog.com"/>
        </foaf:Person>
    </foaf:knows>
</foaf:Person> </rdf:RDF>
```

Exemplo 2.6: Formato RDF/XML

2.2.4 RDFa

RDFa não é um formato de serialização puro para RDF, mas sim um modo de anotar as páginas da Web em formato *eXtensible HyperText Markup Language* (XHTML) com dados RDF. RDFa tem como princípio a necessidade de publicar o conteúdo uma única vez, misturando o conteúdo visualmente legível com o legível e interpretável por máquinas. Esta é uma filosofia similar ao de microformatos, mais simples, com uma abordagem mais *ad hoc* para adicionar anotações semânticas ricas em conteúdo XHTML. RDFa usa um pequeno conjunto de atributos XML que são adicionados a marcas existentes do conteúdo XHTML para especificar a semântica contida na informação que é apresentada. Esses atributos esclarecem o significado semântico de conteúdo existente de XHTML[2, 3].

2.3 Friend of a friend (FOAF)

FOAF significa *Friend Of A Friend* (FOAF). O objetivo deste projeto é criar uma Web em que as pessoas de uma forma ontológica são descritas, assim como as suas atividades e as suas relações. Esta ferramenta pode ser comparada com o Facebook, visto que o princípio é o mesmo. FOAF permite que grupos de pessoas sejam capazes de criar uma rede social sem ser necessário uma base de dados central.

2.4 Web Ontology Language (OWL)

Web Ontology Language (OWL) é uma recomendação do W3C e consiste numa linguagem para definir e instanciar ontologias na Web. A OWL consegue transmitir tudo o que uma ontologia consegue descrever. Uma ontologia OWL pode conter descrições de classes, as suas respetivas propriedades e os seus relacionamentos.

Uma ontologia define formalmente os termos usados para descrever e representar uma área de conhecimento. Numa ontologia, os termos são agrupados como classes semânticas (ou conceitos) e é definido um grupo de indivíduos (instâncias) com propriedades semelhantes. As ontologias incluem definições, conceitos e relacionamentos, devendo permitir a partilha e reutilização do conhecimento[2, 4, 5].

Os recursos são definidos entre si de acordo com um grafo. Essa estrutura permite a automação e manipulação dos dados. A ontologia é semelhante a um dicionário de sinónimos,

tendo uma única diferença, o dicionário de sinónimos conecta conceitos entre eles de acordo com relações específicas: sinónimo, homónimo, hierarquia, termo associado; a ontologia adiciona regras e ferramentas de comparação sobre e entre os termos, grupos de termos e relações: equivalência, simetria, pelo contrário, cardinalidade, transitividade, etc..[6, 7, 8]

Em termos comparativos, o RDF é mais limitado para fazer uma lógica de raciocínio do que uma ontologia, uma vez que é complexo navegar na natureza das relações (por exemplo, nas propriedades inversas)[4, 5].

A OWL foi pensado para o uso de máquina-máquina, tentando oferecer o máximo de autonomia. OWL é uma peça fundamental na Web Semântica. OWL tem três sub-linguagens:

- `OWL`Lite, criada essencialmente para utilizadores que necessitam de uma classificação hierárquica e restrições simples. OWL lite tem a menor complexidade formal.

- `OWL`DL, criada para utilizadores que desejam a máxima expressividade. Esta sub-linguagem inclui todas as construções da linguagem OWL, mas somente podem ser usadas com algumas restrições.

- `OWL`Full, criada para utilizadores que desejam a máxima liberdade e expressividade sintática do RDF. Este tipo de linguagem torna-se impraticável no âmbito computacional. [2, 9, 10].

2.5 Interoperabilidade de Ontologias

Em muitas situações existem fontes distintas de informação, cada uma com a sua ontologia para representar a mesma ideia. No caso de existirem duas ontologias distintas, mas a representarem a mesma ideia, é possível fazer uma junção dessas ontologias. Existem vários mecanismos para efetuar essa junção de ontologias, entre eles: combinação de ontologias e alinhamento de ontologias [11, 7].

2.5.1 Combinação de ontologias

A combinação de ontologias procura agregar as ontologias originais, criando uma só, com todos os termos. Por exemplo, uma ontologia **O1** sabe o que é um **rato** e outra ontologia **O2** sabe o que é um **mamífero**, sendo ambos os conceitos compatíveis. A relação resultante entre estas duas ontologias combinadas numa só, permite inferir que um rato se trata de um mamífero.

2.5.2 Alinhamento de Ontologias

No alinhamento de ontologias, as ontologias originais mantêm-se separadas, mas são adicionadas ligações entre os seus pontos equivalentes. Deste modo, é possível uma ontologia usar informação da outra e vice-versa. O alinhamento de ontologias é normalmente usado em ontologias complementares. Utilizando novamente o exemplo de rato - mamífero (referido na secção anterior), para este caso o resultado seria uma ligação unidirecional de rato para mamífero, visto que rato é um mamífero mas nem todos os mamíferos são ratos [4, 12, 13, 14].

2.6 Simple Protocol and RDF Query Language (SPARQL)

Acrónimo formado por ***S**imple **P**rotocol **a**nd **R**DF **Q**uery **L**anguage* (SPARQL)[15], trata-se de uma linguagem de pesquisa em *query standard* para grafos RDF. SPARQL contém um número de poderosas funcionalidades como, por exemplo, filtrar resultados conseguindo construir novos grafos apenas baseados em *queries*. Em SPARQL existem quatro formas de *queries*: SELECT, CONSTRUCT, ASK e DESCRIBE. No mínimo uma plataforma semântica terá de suportar o SELECT de SPARQL[1].

2.7 Fontes de informação para tecnologias semânticas

2.7.1 Linked Data

Para além de repositórios e registos, existe uma outra abordagem para distribuir o poder das ontologias e dados relacionados. *Linked data* é um projeto que permite que todos os dados sejam ligados entre si. Mais concretamente o objetivo do *Linked data* é permitir a criação de ligações úteis a fontes de dados e ontologias existentes, como é ilustrado na figura 2.2. Estas ligações oferecem a facilidade de encontrar informação útil mas desconhecida, uma vez que os dados se encontram interligados. Bastando seguir as ligações podem-se facilmente encontrar dados interessantes, o que seria impossível numa pesquisa direta pelo facto de não se ter o seu conhecimento [3]. Para isso *Linked Data* apresenta quatro regras fundamentais:

1. Usar URIs como nome dos objetos.

2. Usar *Hypertext Transfer Protocol* (HTTP) URIs para que os nomes e recursos sejam efetivamente localizáveis na Internet.

3. Ao aceder ao URI seja fornecida informação útil.

4. Incluir ligações para outros URIs, para que seja possível encontrar mais informação associada.

Apesar destas orientações exprimirem ideias muito vagas, conseguem despertar interesse entre entidades de diversas áreas como por exemplo: Saúde, Química, etc..

Um grafo global deve ser construído com a união de muitos grafos pequenos de dados distribuídos pela Internet, tal como Tim Berners Lee imaginou[4]. Para mostrar esse tipo de estrutura, uma comunidade para *Linking Open Data* (LOD) emergiu, desenvolvendo práticas recomendadas em torno da publicação de dados semânticos distribuídos. Enquanto RDF fornece formas padronizadas para a serialização de informações, a comunidade de LOD desenvolveu métodos padrão para aceder RDF serializados pela Internet. Esses métodos incluem receitas padrão para descodificar URIs e/ou editores de dados com sugestões sobre a preparação e a implantação de dados na Internet. É igualmente importante a existência de uma comunidade de editores necessários para produzir este grafo global, como um conjunto de aplicações que utilizem estas coleções de dados semânticos distribuídos pela Internet[5] [1, 8].

[3] http://www.w3.org/DesignIssues/LinkedData.html

[4] `http://www.ted.com/talks/lang/pt-br/tim_berners_lee_on_the_next_web.html`

[5] Liked Data Web site: `http://linkeddata.org/`

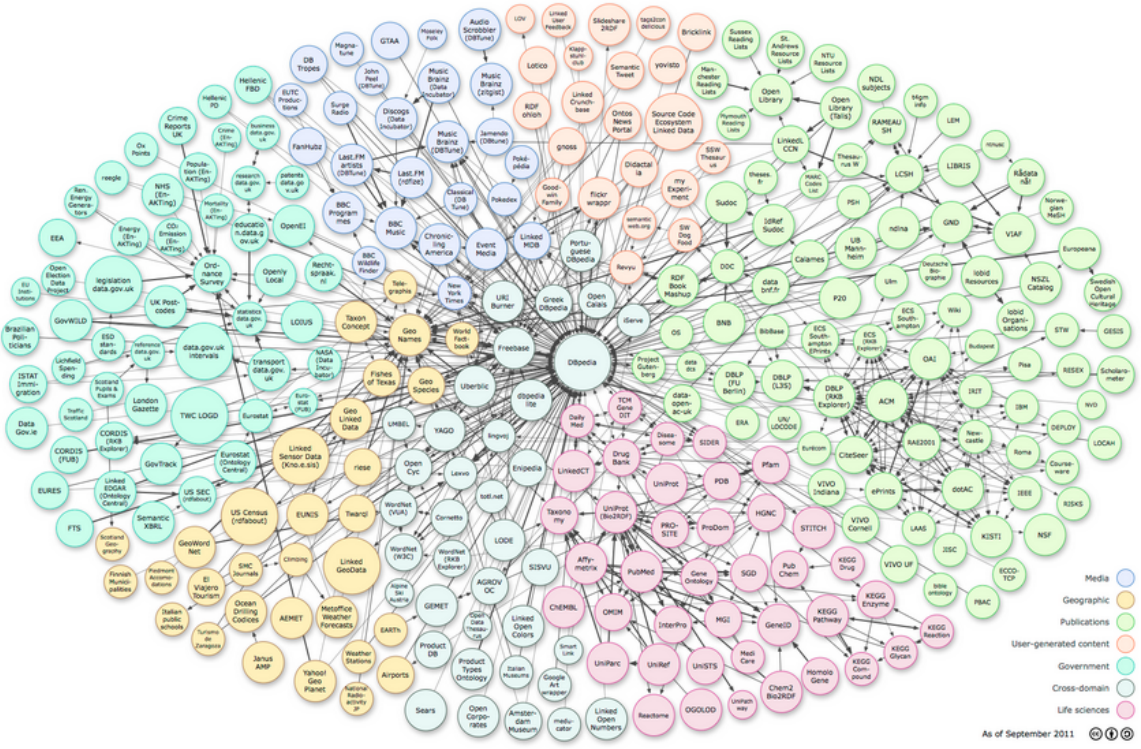

Figura 2.2: LOD cloud diagram [16]

2.7.2 DBpedia

DBpedia é um projeto cujo objetivo consiste em extrair todo o conteúdo existente no Wikipédia[6], mas com a vantagem de ter uma ontologia associada, como também, todos os seus dados interligados, sendo possível a obtenção dos dados num formato RDF , *"DBpedia is the Semantic Web mirror of Wikipédia"*[7]. DBpedia também contém um *endPoint* SPARQL para se poderem obter as informações através de *queries*. Deste modo, é possível colocar questões sofisticadas sobre informações que se encontram na Wikipédia. Opcionalmente, é possível obter ligações com outras fontes de dados (por exemplo *http://www.freebase.com/*) [8].[17, 18]

Este projeto teve inicio na Universidade Livre de Berlim em parceria com a Universidade de Leipzig e com a colaboração do OpenLink Software. Este projeto iniciou no ano 2007 e a sua versão estável surgiu em Agosto de 2012.[19] É um projeto que se encontra disponível sob licenças livres, permitindo a utilização de qualquer utilizador.

Em Setembro de 2011, os dados do DBpedia descrevem mais de 3.64 milhões de "coisas", dos quais 1.83 milhões são classificados numa ontologia consistente, incluindo 416 mil celebridades, 526 mil lugares, 106 mil álbuns de música, 60.000 filmes, 17.500 jogos de vídeo, 169 mil organizações, 183 mil espécies e 5.400 doenças. Todos estes dados são fornecidos em 97 idiomas diferentes.[20]

[6]Wikipédia Web site: `http://www.wikipedia.org`
[7]`http://live.dbpedia.org`
[8]DBpedia Web site: `http://dbpedia.org/`

Este projeto, utiliza RDF para representar a informação extraída. Em setembro de 2011, a base de dados do DBpedia consiste em mais de mil milhões de "peças"de informação (triplos RDF), dos quais 385 milhões foram extraídos da edição de Inglês da Wikipédia e 665 milhões foram extraídas de edições em outros idiomas. [21]

2.7.3 LinkedMDB

LinkedMDB é muito semelhante ao DBpedia. Como o DBpedia contém o conteúdo existente no Wikipédia, LinkedMDB contem o conteúdo existente no IMDB[9] e contém também um *endPoint* SPARQL. [10]

2.8 Frameworks para Web Semânticas

A maior parte das *frameworks* para Web Semântica são uma coleção de ferramentas vocacionadas para criar e manipular conhecimento guardado. Estas *frameworks* têm fundamentalmente três componentes: *storage, access e inference* (figura 2.3).

Storage é onde se encontra o repositório RDF. A componente *Access* é o responsável pelo processamento de chamadas (*queries*) ao repositório ou *Application Programming Interface* (API)s que fornecem e modificam informação. Por último, a componente *inference* é a responsável pelo raciocínio efetuado sobre uma interpretação de uma OWL com um base de conhecimento (*knowledgebase*).

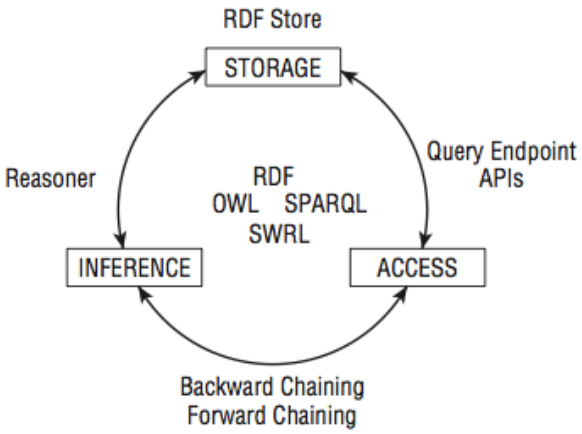

Figura 2.3: Framework semântica consiste normalmente em três componentes fundamentais: *storage, access e inference*. [1]

Basicamente, estas *frameworks* servem para guardar, dar acesso e inferir factos. Estes factos podem ser considerados como explícitos ou implícitos. Os factos explícitos encontram-se diretamente na *triplestore*, enquanto que os factos implícitos surgem da inferência dos factos explícitos com ontologias semânticas e regras de conhecimento provenientes da *triplestore*[2].

[9]IMDB Web site: http://www.imdb.com/
[10]LinkedMDB Web site: `http://www.linkedmdb.org/`

2.8.1 RdfLib

RdfLib[22] é uma *framework* bastante sólida e extensiva para python. Esta *framework* contem *parsers* e *serializers* para RDF/XML, N3, NTriples, Turtle, TriX and RDFa. Esta biblioteca oferece uma interface para manipular um grafo, que pode ser armazenado através da memória, MySQL, Redland, SQLite, Sleepycat, ZODB e SQLObject.

A última versão é RdfLib 3.0[11], mas a mais usada ainda é a versão 2.4[12]. Existe uma grande diferença entre a 3.0 e a 2.4. A maior das diferenças reside no fato de algumas das bibliotecas existentes na versão 2.4 terem sido separadas (denominado rdfextras[13]).

2.8.2 RdfAlchemy

O objetivo do RdfAlchemy[23] é permitir que qualquer pessoa ao utilizar Python tenha acesso a uma triplestore RDF através da sua API. RdfAlchemy é um *Object RDF Mapper* (ORM), pode também funcionar como outras framework como, por exemplo, rdflib, Sesame, e Jena. O suporte para SPARQL está presente, mas ainda se encontra num estado muito instável.

2.8.3 FuXi

O Fuxi[24] foi desenvolvido para Python. É um sistema lógico bi-direcional de raciocínio para a Web Semântica. Necessita da RdfLib 2.4.1 ou 2.4.2 e não é compatível com a versão 3 (esta diferença foi explicada na secção 2.8.1). FuXi oferece uma solução completa para representação de conhecimento e raciocínio sobre a Web Semântica; é bastante sofisticado e bem documentado (em parte através de vários artigos académicos). A desvantagem é a acentuada curva de aprendizagem necessária à sua utilização.

2.8.4 Sesame

Sesame é um projeto *open source framework* para consultar e guardar dados RDF. Foi desenvolvido inicialmente pela empresa holandesa Aduna como um protótipo de investigação para a União Europeia denominado *On-To-Knowledge*. Atualmente, tem sido desenvolvido como um projeto comunitário[14]. Sesame tem uma excelente interface de administração incluída na distribuição principal, é fácil de instalar e tem um forte desempenho[1].

2.8.5 Apache Jena

Jena é uma *open source framework* para Java proveniente do laboratório da Web Semântica Hewlett-Packard's e também de pesquisadores dos laboratórios da HP no Reino Unido. O seu primeiro lançamento foi em 2001 e desde então tem sido amplamente utilizada. Em 2008 Jena deixou a HP, passando em 2009 a projeto incubado na Apache[15].

Jena é idêntica ao Sesame ao nível das funcionalidades que oferece, no entanto, as APIs da Jena são mais completas pois suportam OWL. Contém várias regras diferentes para *reasoners*, suporta SPARQL e vários tipos de armazenamento em forma de grafos. Para além disso, com

[11] http://code.alcidesfonseca.com/docs/rdflib/index.html
[12] http://www.rdflib.net/rdflib-2.4.0/html/index.html
[13] http://code.google.com/p/rdfextras/
[14] http://openrdf.org
[15] Jena Web site: `http://incubator.apache.org/jena/`

Jena também se tem acesso a uma API RDF, sendo possível a leitura e escrita em RDF/XML, RDF/XML-ABBREV,N-Triples, N3 e Turtle[13, 10].

Jena contém todos os conceitos RDF: recurso, propriedades, literais e *statements* (Tripletos: <suj pred obj>). Deste modo, é possível ter vários *statements* relacionados, obtendo assim um modelo representativo de um grafo RDF. Além destas possibilidades, Jena oferece um leque variado de interfaces para a manipulação dos modelos.

Jena é projetado de modo a que os motores de inferência possam ser ligados a modelos, e, através disso, seja feito um raciocínio. Jena tem algumas ferramentas de inferências, mas mesmo assim ainda tem capacidade de automatismo muito limitado. Depois de obter um objeto *reasoner*, este precisa de ser ligado a um modelo. Para isso é necessário modificar as especificações do modelo para não seja somente uma adição do objeto ao modelo. Sendo assim, é necessário que as especificações fiquem coerentes e não repetidas.[25]

Devido a sua origem como projeto de investigação, Jena tem uma ampla variedade de novas bibliotecas, a maioria experimentais. A desvantagem de Jena é que tende a ser mais complexa em relação ao Sesame, sendo necessário igualmente uma curva de aprendizagem bem mais acentuada.[1, 2].

O Jena tem uma série de soluções de armazenamento RDF, uma delas é *Triple Data Base* (TDB). TDB é um componente de Jena de armazenagem RDF e de consulta. Suporta a gama completa de APIs de Jena. TDB pode ser usado como um armazenamento de alto desempenho RDF numa única máquina. Uma TDB pode ser acedida e gerida com os *scripts* de linha de comando fornecidos e por meio da API de Jena[13]. Um dataset TDB é armazenado num único diretório no sistema de ficheiros. Um conjunto de dados consiste numa tabela de nós, Triple e Quad, índices e Tabelas de prefixos.

TDB é otimizado para triplestores. Tem a capacidade de armazenamento de um grande número de tripletos, com eficiência, sem requerer uma base de dados relacional adicional. Tem um alto desempenho e escalabilidade, sendo uma base de dados dedicada a grafos RDF. Tem a vantagem de ter uma escrita rápida, de ter um custo reduzido na conversão do *nóId* para nó. Os *nóIds* podem ser menores que as suas *hashes*, reduzindo o tamanho dos seus índices. Tem também a vantagem de não ser necessário converter entre SPARQL e SQL. No entanto, TDB não tem nenhum mecanismo eficiente de transações necessitando de um espaço de disco considerável. Uma vez que o mapa nó/*nóId* pode ser muito extenso e apresentar-se apenas como um único ficheiro de indexação, quando há uma transação ou por alguma razão suceder-se inesperadamente um falha na escrita o ficheiro fica corrompido. TDB não tem uma forma de fazer o *rollback* podendo recuperar o ficheiro, de índices, desta forma será impossível uma nova operação com a triplestore. Outra desvantagem é a conversão de URIs/literais necessitar de muitas transações criando custos elevados de acessos ao disco, dependendo da quantidade de informação que se pode guardar na memória volátil[26].

Em TDB existem três índices compostos (árvore B+): *Subject - Predicate - Object* (SPO), *Predicate - Object - Subject* (POS) e *Object - Subject - Predicate* (OSP). Os *nóIds* são identificadores de 64 bits, 8 bits servem para descrever o tipo de nó, os restante 56 bits são referentes ao endereço em disco (figura 2.4)[26, 27].

2.9 Aplicações que usufruem da semântica

As tecnologias semânticas estão a começar a dar resultados e dentro destas tecnologias já começa a surgir algum interesse, desde: empresas, agências governamentais, comunidade

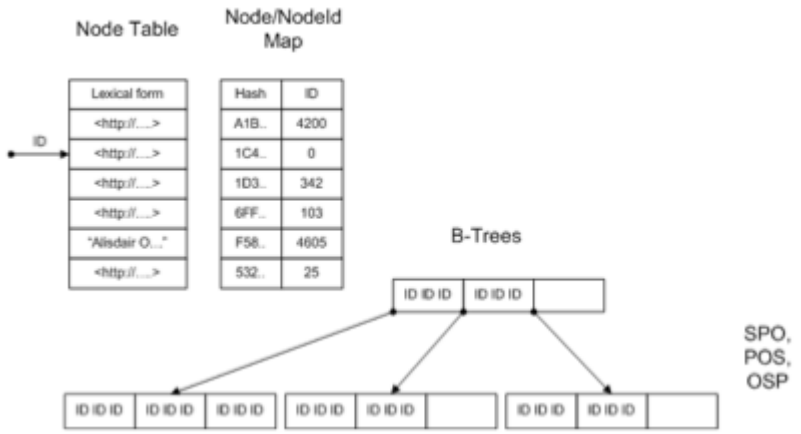

Figura 2.4: Arquitetura TDB

científica, os programadores de aplicações (investigação). A sua incorporação é defendida nos sistemas de informação. A razão fundamental que está a conduzir para este movimento vem do sentimento bastante difundido de nos estarmos a afogar em dados e que o nível de maturidade a que estas tecnologias estão a chegar é suficiente para produzir resultados. Alguns especialistas, como Richard MacManus, fundador da *ReadWriteWeb*[16], vêm estas tecnologias como as maiores tendências para os próximos anos[17].

2.9.1 Tabulador

O tabulador é uma extensão para os *browsers*. Esta extensão é ativada quando o navegador recebe um arquivo RDF e exibe-o de uma forma de fácil perceção. Com o tabulador é possível gerar um grafo através de fontes de dados como o Linked Data. O tabulador permite a navegação através do grafo de uma maneira útil e conveniente.

Além destes grafos, o tabulador pode exibir vários modos de visualização com base em simples tabelas, mapas, calendários, cronogramas e qualquer outro tipo de dados que seja possível criar. Sendo necessário o uso extensivo do *Gleaning Resource Descriptions from Dialects of Languages*[18], A figura **??** ilustra a ontologia de *FriendTracker* através do tabulador[2, 8].

2.9.2 Verbetes SAPO

Verbetes é um produto do SAPO desenvolvido pelo Labs Sapo Up[19]. Esta tecnologia responde a pedidos *WhoIs* para personalidades da vida pública. A semântica nesta ferramenta encontra-se no facto de conseguir recolher de uma forma autónoma informações de personalidades públicas em fontes noticiosas[28].

[16] `http://www.readwriteweb.com/`

[17] `http:///www.readwriteweb.com/archives/10_future_web_trends.php`

[18] ver http://www.w3.org/TR/grddl/)

[19] Este serviço atualmente é gratuito sendo necessário apenas a *key* para o uso da API neste endereço: `https://store.services.sapo.pt/en/Catalog/other/free-api-information-retrieval-verbetes/Purchase/`

2.9.3 BBC Sports

A BBC é talvez uma das maiores empresas que valoriza e explora o poder da semântica na Web. Uma prova disso é a página das notícias de desporto[20], alterada recentemente para suportar a introdução de notícias de forma automática, através de um modelo relacional e de uma *framework* estática de publicação. Por outras palavras, estas notícias são introduzidas somente por máquinas, não existindo nenhuma ação humana envolvida. Para isso, a informação é obtida através de bases de dados de eventos desportivos que se encontram em constante atualização. Por exemplo, existiam servidores nos jogos olímpicos de Londres somente dedicados para esta tarefa. Estes servidores continham uma base de dados que era atualizada com os resultados obtidos pelos atletas.

O resultado é uma arquitetura dinâmica de publicação denominada por *Dynamic Semantic Publishing* (DSP). O *site* da BBC usado para a cobertura dos jogos olímpicos de 2012[21] foi também baseado nesta tecnologia e potenciado pela Web Semântica.

O DSP usa como base a tecnologia *Linked Data* para automatizar as agregações, publicar e redefinir os objetos inter-relacionados de acordo com uma ontologia modelada por eles próprios. Deste modo, é obtido uma ótima experiência e um alto nível de envolvimento com o utilizador.

A BBC desta forma consegue demonstrar o futuro de todas as páginas Web, tendo num site Web o poder de encontrar toda a informação num mesmo local, sem ser necessário efetuar pesquisas noutras fontes de informação. O sistema encarrega-se de pesquisar e apresentar a informação recolhida[29].

2.9.4 Gephi

Gephi é uma ferramenta gratuita e *open source*[22] que gera um grafo ou uma rede sobre um determinado tema à escolha do utilizador. Através da semântica, Gephi vai construindo a sua rede, sendo uma ferramenta ideal para trabalhar com conjuntos de dados complexos porque Gephi produz um grafo com bons resultados visuais. Fornece ainda várias ferramentas para a exploração interativa e para a interpretação dos grafos.[30, 31].

2.9.5 Pesquisa Google

A conhecida Google também alterou muito recentemente o seu algoritmo de pesquisa[32]. A Google sabe que atualmente é impossível pesquisar somente por cadeias de caracteres, eles próprios informam que "coisas"não se refletem em palavras, *"things, not strings"*.

Deste modo, a Google criou um grafo de conhecimento que permite a procura de "coisas", pessoas, lugares entre outros. Devido à Google conter um volume muito considerável de dados, basta um pequeno *upgrade* para pesquisas semânticas serem efetuadas. Bastando unir marcos, celebridades, cidades, equipas de desporto, edifícios, características geográficas, filmes, objetos celestes, obras de arte e muito mais entre eles, podendo assim obter instantaneamente informações relevantes. O grafo de conhecimento da Google não está apenas enraizado em fontes públicas tais como Freebase, Wikipédia e o CIA World Factbook. Ele também é aumentado numa escala muito maior porque estão focados numa ampla abrangência e profundidade. Atualmente a Google contêm mais de 500 milhões de objetos[32], bem como

[20] `http://www.bbc.co.uk/sport/0/`
[21] `http://www.bbc.co.uk/2012/`
[22] `https://gephi.org/`

mais de 3,5 mil milhões de factos e relações sobre esses diferentes objetos. Esta ferramenta é enriquecida com base no que as pessoas pesquisam e no que se pode encontrar na Web.

A própria Google ao anunciar a sua nova forma de pesquisa nomeia três grandes melhorias que a semântica oferece:

1. **Encontrar o que realmente procuramos.**
   Procurar, por exemplo, Taj Mahal monumento ou Taj Mahal o músico: A Google saberá qual será a diferença podendo assim restringir os resultados da pesquisa. Desta forma, a Google enriquecerá o seu grafo de conhecimento conseguindo cada vez mais responder de uma forma acertada.

2. **Obter o melhor resumo.**
   Com o grafo de conhecimentos, a Google consegue compreender melhor a sua consulta. Deste modo, tem o poder de resumir o conteúdo relevante em torno do que é pesquisado e, além disso, este resumo será adaptado para o utilizador que fez a pesquisa. As pessoas podem pesquisar pela mesma "coisa"mas com objetivos diferentes. Um exemplo é uma pesquisa por José Saramago: a primeira informação que surge será, por exemplo, a sua biografia desde a data de nascimento até à data em que faleceu, quais as suas obras, etc... Se numa pesquisa subsequente, se pesquisar por Harol Pinter, uma das conclusões possíveis é que o utilizador está a pesquisar informações sobre laureados com o Nobel de literatura. Assim, é possível ao motor de pesquisa apresentar um resumo mais vocacionado sobre a área em questão. Como a Google está presente em tantas aplicações, o poder de conhecimento é de tal forma grande ao ponto de ter muita informação pessoal, facilmente a ferramenta de pesquisa deles se adaptará ao que se procura.

3. **Pesquisar informação de modo mais profundo e amplo.**
   Conhecimentos inesperados e úteis podem encontrar-se pelo facto dos dados se encontrarem relacionados. Uma pesquisa na Web terá resultados sobre o assunto que se procura, como também informação adicional relacionada como o mesmo. Esse conteúdo relacionado pode ser completamente desconhecido. Sendo assim, é possível a pesquisa de conteúdos desconhecidos de uma forma muito simples, bastando apenas encontrar relações. Imaginando que se deseja o nome especifico de um ator, mas somente é conhecido o nome de um filme que ele representa, com esta ferramenta ao pesquisar pelo nome do filme obtém-se rapidamente o nome do ator.

   Este projeto da Google ainda é um pequeno passo mas uma grande *startup* para a Web 3[1, 32].

2.10 Plataformas de gestão de conhecimento

2.10.1 OObian

OObian é uma plataforma de gestão de conhecimento feita especialmente para ambientes empresariais. Esta plataforma fornece uma melhor pesquisa e navegação através do conhecimento em comparação com Jena TDB. OObian extrai e indexa informações de dados estruturados e não estruturados difundidos dentro e fora da organização, transformando-as em conhecimento de negócio real. Em ambientes de trabalho, OObian facilita aos funcionários

encontrar facilmente as pessoas indicadas para a conclusão do seu trabalho. Uma pesquisa produtiva normalmente aumenta a eficiência de um funcionário, ao permitir a sua conexão a um amplo conjunto de pessoas que, por sua vez, o conecta a um vasto conjunto de informações úteis para a realização das suas tarefas.

Através do OObian as pesquisas são feitas de um forma simples e intuitiva, oferecendo uma maior velocidade, como também uma facilidade de compreensão ao nível visual para um utilizador comum que está a tentar encontrar um determinado conteúdo.

Além de utilizar semântica, esta ferramenta usa também *Natural language processing* (NLP), para relacionar informação e perceber de que objeto especifico se trata no contexto da sua utilização[33].

2.10.2 Apache UIMA

Apache UIMA[23] é uma aplicação de gestão de informações não estruturadas, é uma implementação *open source* licenciada pela Apache obedecendo à especificação da UIMA (cuja especificação está a ser desenvolvida simultaneamente por um comité técnico no *Advancing Open Standards for the Information Society* (OASIS)).[24]

São sistemas de software que analisam grandes volumes de informação não estruturada, para obter conhecimentos relevantes para um utilizador final. Com esta aplicação pode-se inserir texto sem formatação e identificar entidades, tais como pessoas, lugares, organizações ou relações.

UIMA permite que as aplicações sejam decompostas em vários componentes, por exemplo:

- Identificação da linguagem.
- Segmentação específica do idioma.
- Deteção de limite da frase.
- Deteção de entidade (pessoa/localizações etc.).

Cada componente contém interfaces definidas pela estrutura e fornece metadados auto-descritivos através de arquivos XML do descritor. Essa estrutura gere esses componentes e o fluxo de dados entre eles. Os componentes são escritos em Java ou C++, os dados que fluem entre os componentes são projetados para mapeamento eficiente entre essas línguas.

UIMA permite encapsular componentes como serviços de rede. Conseguindo ser escalável mesmo com uma quantidade significativa de dados, UIMA suporta um sistema com vários nós de processamento, conseguindo replicar o processamento sobre um *cluster* existente na rede.

Existem vários componentes externos que podem ser adicionados, como Consumidores e Anotadores (*Annotator components*)[25]. Estes Consumidores e Anotadores têm como função analisar as informações não estruturadas, como por exemplo, podem usar expressões regulares ou dicionários. Os utilizadores têm a possibilidade de criar os seus próprios anotadores[26] ou então podem usar os anotadores já disponíveis (de raiz ou em repositórios online[27]), sendo apenas necessária a sua configuração.

[23] `http://uima.apache.org/`

[24] `http://www.oasis-open.org/committees/download.php/28492/uima-spec-wd-05.pdf`

[25] Anotadores existente: `http://uima.apache.org/sandbox.html#uima-addons-annotators`

[26] Como criar um anotador: `http://uima.apache.org/doc-uima-annotator.html`

[27] Repositórios externos: `http://uima.apache.org/external-resources.html`

Capítulo 3

Arquitetura

Do ponto de vista de arquitetura de software, este projeto pode ser dividido em três módulos principais: Aquisição de dados, Interface de Acesso Programático e Visualização de informação.

3.1 Aquisição de dados

Este módulo é o responsável pela introdução de novos dados. Numa fase inicial do seu desenvolvimento, a opção de *parsing* a páginas Web foi ponderada. Devido ao IMDB ser dos sites mais completos e credíveis ao nível cinematográfico e não conter nenhuma API disponível para se obter os dados, foi pensado fazer *parsing* ás suas páginas Web. No entanto, por razões legais, o *parsing* do IMDB não é permitido e, por isso, foi feito um estudo de páginas igualmente credíveis para a obtenção de dados. No entanto, dada a volatilidade desta forma de aquisição de dados em função da apresentação dos mesmos, rapidamente esta hipótese foi desconsiderada.

A alternativa foi o acesso a bases de dados semânticas. Com o *parsing* seria necessário um processamento de linguagem muito avançado para conseguir construir uma base de dados com as propriedades ontológicas que estão disponíveis numa *triplestore* como, por exemplo, a que se encontra disponível no DBpedia. Desta forma, com uma *triplestore* bem implementada é possível obter respostas a perguntas (*queries*) muito minuciosas como por exemplo: "Quantos filmes de comédia romântica de Hollywood são dirigidos por uma pessoa que nasce numa cidade cuja a temperatura média se encontre acima de 15 graus?".

Antes da descrição de como é efetuada a aquisição dos dados é necessário salientar que, quando foi desenhada esta arquitetura as informações apresentadas pelo DBpedia eram muito escassas. Desta forma, o projeto foi desenhado e desenvolvido para obter informações em mais *sites* para além do DBpedia. Estas informações são adquiridas em fontes que, atualmente, são fornecedoras de dados ao DBpedia.

No entanto, com o crescimento do DBpedia a informação que no início não se encontrava, com o passar do tempo passou a existir. Desta forma, surgiam dados duplicados. Sendo assim, houve a necessidade do projeto ser redesenhado para eliminar a redundância. Esta redundância foi corrigida, no entanto, com a evolução constante do DBpedia não há garantia que surja novamente, porque ainda existem dados provenientes de fontes externas ao DBpedia.

O método responsável por iniciar todo o ciclo de armazenamento deste projeto foi o *title provider*. A sua principal funcionalidade consiste em encontrar URIs de filmes e de séries em

vários *endpoints*.

Inicialmente foi criada uma máquina de estados cíclica (fig.3.1) de forma a estar em constante funcionamento. Quando este módulo fosse executado a primeira operação seria a aquisição de dados através do EPG devido à sua elevada prioridade. O EPG do SAPO é um serviço onde se obtém a programação televisiva. Esta prioridade justifica-se pelo facto da informação que o projeto deve ter como disponível, seja a informação associada a toda a programação que, atualmente ou num espaço curto de tempo, irá decorrer na televisão.

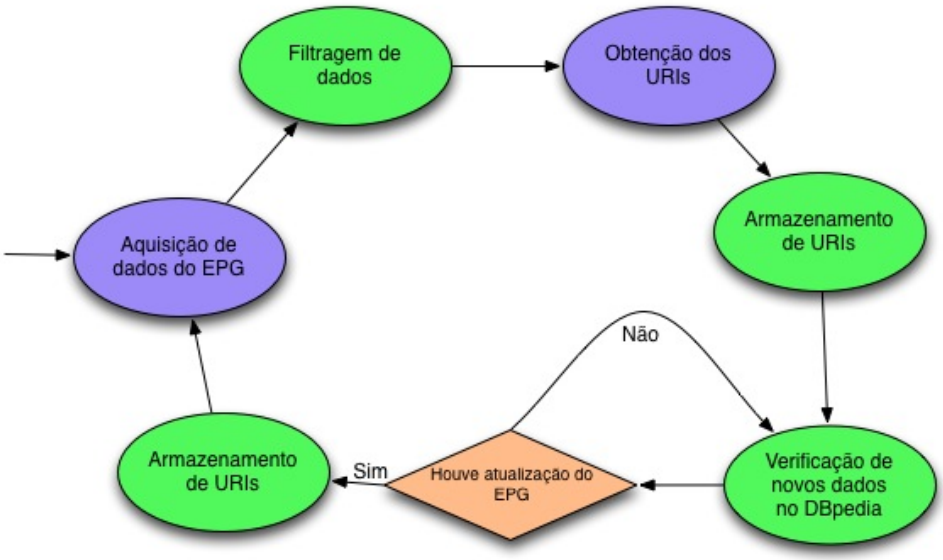

Figura 3.1: Primeira máquina de estados do módulo de captura implementada

Depois de obtida toda a informação correspondente ao EPG, o programa entraria num segundo estado da máquina de estados, a filtragem de dados. Neste estado é realizada uma filtragem de todos os resultados encontrados, de forma a obter uma lista de títulos não repetidos de todos os eventos de cada canal.

Depois de adquirir a lista de todos os títulos segue-se para o estado seguinte, a obtenção dos URIs correspondentes. Neste estado são feitos *queries* ao *endpoint* SPARQL do DBpedia. Se o filme ou a série se encontrar na DBpedia será obtido o respetivo URI. Esse URI será guardado numa lista para posteriormente ser enviado para o segundo estado. Quando é concluída a leitura do EPG, é efetuada uma leitura ao DBpedia, verificando se todos os URIs que nele se encontram já se encontram armazenados localmente. No caso da existência de um URI que não exista na triplestore é introduzido na lista onde se encontram os URIs que serão enviados para o segundo estado.

Após a implementação inicial desta máquina de estados, verificou-se que para o propósito exigido era muito lenta. Teria informação excessiva e muita dessa informação não seria utilizada. Por outro lado, era feito um *query* SPARQL para obter todos os URI's do DBpedia. Este *query* demorava muito tempo e não bastava somente um *query* para esse fim. A API do DBpedia apenas responde com dez mil URIs no máximo, sendo necessário criar um *query* com um *offset* de oito mil resultados podendo, assim, conseguir todos os URI's correspondentes a filmes e a séries existentes no DBpedia. Praticamente após a conclusão desta máquina de

estados a DBpedia lançou um serviço de *lookup*[1]. Através deste serviço são obtidos resultados muito mais rapidamente em comparação ao tempo dos *queries* SPARQL ao *end point* do DBpedia (este serviço será explicado posteriormente com mais detalhe). Sendo assim, para além da fraca performance obtida por parte da máquina de estados e face ao surgimento deste novo serviço, optou-se por coloca-la de parte.

Pela razão dos projetos dependentes a este necessitarem unicamente de dados que se encontram no EPG, é irrelevante obter conteúdos que não se encontram nele. Por esse motivo, optou-se por apenas usar o EPG como *"trigger"* do projeto. Através dele são extraídos todos os nomes de filmes ou séries que se deseja obter informação.

Depois de obtida a lista de nomes através do EPG é realizada uma pesquisa no DBpedia verificando a existência desse nome. Caso exista informação coligada a esse nome é solicitada toda a informação correspondente ao mesmo. Esta informação obtida pelo DBpedia é guardada no seu formato original numa TDB local.

Para um correto armazenamento, a TDB deve também guardar a onotologia associada aos objetos que irão ser armazenados. Inicialmente pensou-se em criar uma ontologia própria para este projeto. Para tal, foi criado um módulo cuja função seria criar e adicionar propriedades à ontologia na base de dados local (descrito no capítulo seguinte). Para isso, é criado um novo URI para cada nova propriedade com o "hostname" específico do servidor. Desta forma, não teria uma ontologia tão elaborada e extensa cujo fim seria descrever todos os objetos existentes no mundo, mas sim uma ontologia em que somente descreva filmes e séries. Por outro lado, devido aos dados obtidos serem fundamentalmente do DBpedia, como também, pela facto de um filme poder referir tudo o que exista no mundo, obrigando assim, à ontologia associada a esse filme ter que conseguir referir igualmente o mundo, parte da ontologia do DBpedia não seria suficiente. Estas razões levam que a ontologia parcial seja posta de parte, sendo coerente que a ontologia usada seja a mesma do DBpedia.

Os géneros são obtidos através do DBpedia e complementados com os géneros encontrados no Freebase[2]. Estes géneros são posteriormente processados de forma a apenas armazenar géneros pré-definidos. Estes géneros foram previamente definidos entre todos os projetos que irão partilhar esta *triplestore*.

Para a obtenção da cotação dos filme é usado o *rotten tomatoes*[3].

Um dos problemas que surgiu foi o facto de existirem filmes/séries com nomes idênticos. Por exemplo, através do DBpedia, ao pesquisar pelo nome "Titanic"obtêm-se cinco resultados distintos. Este problema surge pelo facto do EPG apenas fornecer o nome do filme/série. Para a sua resolução foram estudadas várias metodologias para apurar entre os filmes ou séries com nomes idênticos, o mais conhecido. Verificou-se que os primeiros resultados da pesquisa no Google do nome de um filme/série, correspondem ao mais conhecido. Por exemplo, se se pesquisar por "Titanic"no Google, o primeiro filme que surge é o "Titanic"de 1997, que é precisamente o mais conhecido. Verificou-se, também, que através da cotação do *rotten tomatoes* os filmes que têm maior cotação são os mais conhecidos. O *rotten tomatoes* disponibiliza dois tipos de cotação: a dos críticos de cinema e dos utilizadores do site. No decurso deste trabalho foi apenas considerada a cotação dada pelos utilizadores. Ainda assim, de acordo com o exemplo referido anteriormente, no *rotten tomatoes* surgem dois filmes "Titanic", o de 1997 e o de 1953, ambos com a mesma cotação. Sendo assim, seria necessário encontrar outro

[1]http://wiki.dbpedia.org/lookup/
[2]Freebase website: http://www.freebase.com
[3]Rotten tomatoes website: http://www.rottentomatoes.com

método para destingir os filmes. Para isso, é escolhido o filme que aparece em primeiro lugar com a mesma cotação. Esta solução foi testada múltiplas vezes, o que permitiu concluir que o *rotten tomatoes* disponibiliza os filmes por uma ordem crescente, do mais ao menos conhecido. No entanto, não é comum encontrar dois filmes com o nome e cotação igual. Na tabela 3.1, que apresenta uma lista de alguns filmes testados e que apresentam o mesmo nome, apenas o filme "Titanic"e o "12 Angry Men"contêm a mesma cotação.

Title	Primeiro Resultado	Cotação	Segundo Resultado	Cotação
Titanic	Titanic(1997)	87%	Titanic(1953)	87%
Psycho	Psycho(1960)	97%	Psycho(1998)	38%
Godfather	The Godfather (1972)	100%	The Godfather, Part II (1974)	98%
12 Angry Men	12 Angry Men (1957)	100%	12 Angry Men (1997)	100%
Seven Samurai	Seven Samurai (1954)	100 %	The Seven Samurai (2011)	0%
City of God	City of God(2002)	90%	City of God (2011)	0%

Tabela 3.1: Resultados através do *Rotten Tomatoes*

Um outro problema na aquisição do filme/série é o facto do EPG ser em português e, como consequência, os títulos encontram-se em português. Como não existe mais nenhum dado relevante que torne possível obter o título original, e como também nos filmes/séries as traduções não são literais (impossibilitando uma simples tradução), a solução encontrada foi usar a API do Freebase para se obter a tradução do título do filme/série. Deste modo, é feita uma pesquisa pelo Freebase em vez de usar o *endpoint* SPARQL do DBpedia para encontrar o título original porque a resposta é muito menor através do Freebase comparativamente a um *query* SPARQL (tempos são ilustrados no próximo capítulo).

Para um filme/série que contenha o nome idêntico, o Freebase devolve um conjunto de filmes ou séries, visto que apenas com a informação do título não consegue saber qual das traduções pertencem ao filme/série pedido. Novamente, a solução implementada é exatamente a mesma que a do *rotten tomatoes*: os filmes ou séries devolvidos pela API do Freebase encontram-se por ordem crescente de popularidade, basta apenas escolher o primeiro.

Um outro fator constatado ao longo deste projeto corresponde ao facto do DBpedia ter informação errada no que corresponde a traduções de títulos. Todos os filmes contém uma propriedade (rdfs:label) que contêm vários literais, todos são acompanhados pela linguagem correspondente, por exemplo *Titanic (1997)@pt*. Mas a migração para o DBpedia ocorreu de forma faseada, sendo introduzido o nome em inglês em todos os campos de línguas distintas. Por exemplo, a tradução do filme "Inception"para português é "A Origem"e no DBpedia surge *Inception@pt* em vez de *A Origem@pt*. Desta forma, é impossível obter alguma coerência nas traduções como também não é garantido sucesso na pesquisa pelo nome em português.

Para além dos dados adquiridos através do DBpedia, são também usadas mais fontes de dados para poder enriquecer com mais informação o que se está caracterizar. Uma das fontes é o LinkedMDB, já referido na secção 2.7.3. Apesar do LinkedMDB ser uma das fontes primárias ligadas ao DBpedia, no início deste projeto encontrou-se informação útil no LinkedMDB que ainda não se encontrava no DBpedia. Por esse motivo, todos os atores de um determinado filme/série seriam também procurados no LinkedMDB. Esses atores apenas seriam adicionados à triplestore no caso de eles ainda não existirem no DBpedia.

Posteriormente, o DBpedia foi migrando muito lentamente informação do LinkedMDB. Assim, em alguns filmes/séries já se encontrava a totalidade dos atores no DBpedia, em outros casos, apenas se encontrava uma parte desses atores em comparação com o LinkedMDB. Uma vez que este problema se prolongou durante algum tempo, foi criado um novo método que importava atores que não existissem no DBpedia, do LinkedMDB.

Para encontrar o filme idêntico ao do DBpedia no LinkedMDB, são efetuados os mesmos procedimentos anteriormente indicados na obtenção do URI através do DBpedia. LinkedMDB, tal como o DBpedia, contém um *endpoint* SPARQL, através do qual, é feito um *query* para obter todos os filmes/séries com o nome idêntico ao filme/série encontrado no DBpedia. Depois de obtidos vários resultados, é comparado o ano e o realizador do filme/série, garantindo tratar-se do mesmo filme/série obtido através do DBpedia e do LinkedMDB.

Esta implementação foi desativada no final do mês de Julho, uma vez que toda a informação do LinkedMDB foi migrada por completo para o DBpedia[4].

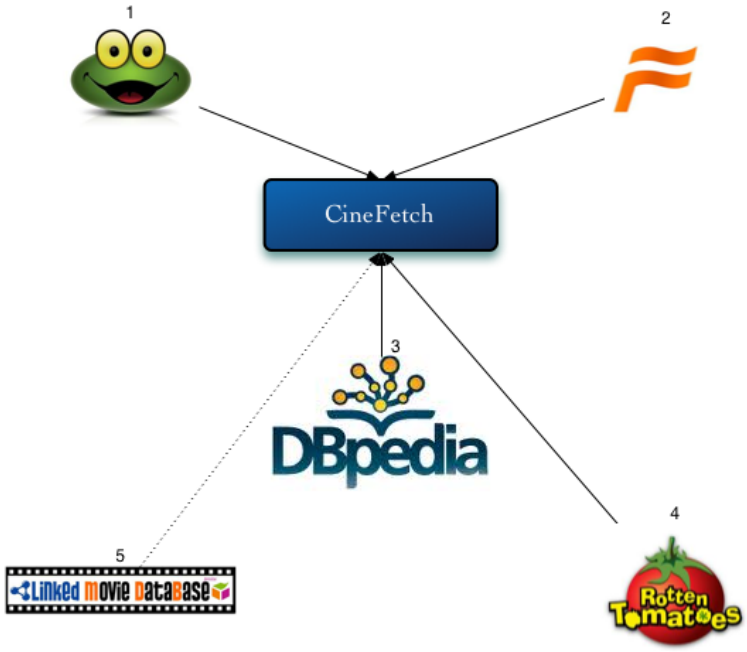

Figura 3.2: Aquisição de dados

O esquema ilustrado na figura 3.2 demonstra como é feita a aquisição de dados neste projeto.

Para a obtenção dos dados, este projeto efetua os seguintes passos:

- Obtenção do título do programa televisivo através do EPG do SAPO(1);
- Verificação da existência no DBpedia do título obtido(3);
- Caso de não existir nenhum resultado no DBpedia;
  - É feita a obtenção do título original através do Freebase(2);

[4]Informação obtida através da *mailing list* dbpedia-discussion-request@lists.sourceforge.net

- Obtenção da informação no DBpedia relativa ao título em questão(3);
- Obtenção da classificação do título em questão(4);
- Obtenção, para o filme em questão, dos géneros a ele associados no Freebase (2), complementados com os géneros a ele associados no DBpedia (3);
- Verificação da existência de atores no LinkedMDB(5);
- Caso não existam atores no DBpedia, mas sim no LinkedMDB ;
  - Obtenção dos atores(5);

3.2 Application Programming Interface (API)

Este projeto contém APIs através da qual é possível obter dados de um filme ou série, bem como introduzir dados adicionais. No início deste projeto não foi contemplada a criação de APIs. No entanto, com a evolução dos outros projetos que se encontravam dependentes deste, foi necessário criar várias APIs para serem efetuados testes. Foram criados os seguintes pontos de acesso *Representational State Transfer* (ReST), acessíveis por GET: Explorador, pesquisa pelo nome, Visualização de um objeto e armazenamento.

Explorador: através desta API é possível visualizar e explorar a triplestore através do *browser*. Desta forma, é adquirida uma melhor compreensão dos dados e da estrutura da triplestore.

Pesquisar pelo nome: asta API, fornece um ficheiro XML com os parâmetros requeridos pelos projetos dependentes. Somente pode ser pedido um único objeto de cada vez.

Visualização de um objeto: ao colocar o URI do objeto no *browser* é retornado num formato *HyperText Markup Language* (HTML) todo o conteúdo do objeto que é pesquisado. Desta forma, torna-se possível a visualização de todo os correspondente ao objeto.

Armazenamento: caso seja necessário, esta API permite ao utilizador introduzir manualmente um objeto na triplestore.

3.3 Visualização de Informação

Apesar de não ser um dos objetivos deste projeto, foi elaborado um método que permite a visualização dos objetos que estão armazenados na *triplestore*. Este método gera um grafo, colocando um objeto previamente selecionado como o nó raiz do grafo e, automaticamente, cria todas as ligações subjacentes nos seus níveis correspondentes. Na figura 3.3 encontra-se um exemplo em que o nó raiz é o filme "Pulp Fiction", no primeiro nível encontram-se todos os objetos diretamente ligados a esse nó raiz (realizador, *abstract*, ano de realização, entre outros). Os atores na *triplestore* encontram-se diretamente ligados ao filme. No grafo gerado, os atores encontram-se todos ligados a um nó denominado como *Atores*. Este nó trata-se de um *blank node* (explicado no capítulo 2.2) melhorando assim a forma de ligação dos nós, obtendo uma melhor compreensão do grafo.

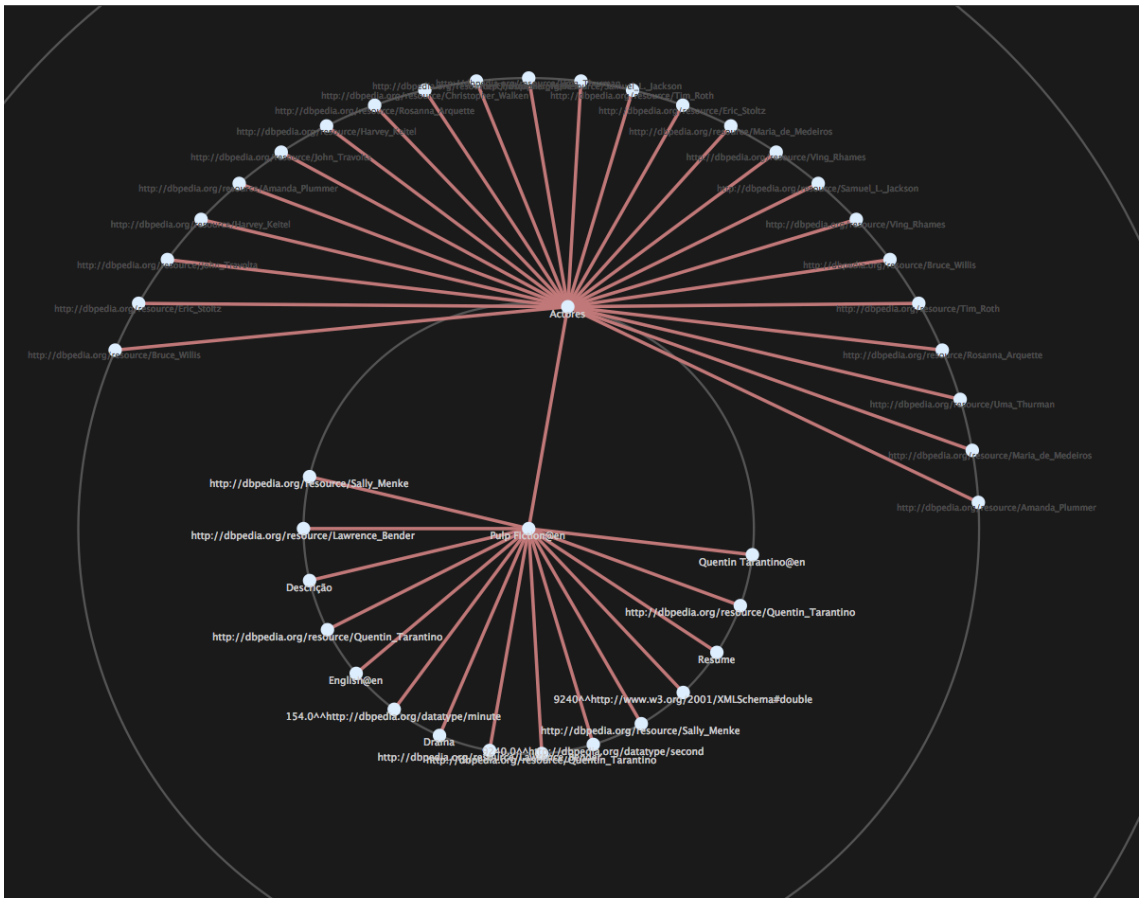

Figura 3.3: Visualização do filme *Pulp Fiction*.

Capítulo 4

Implementação

Este capítulo aborda alguns dos aspetos mais relevantes da implementação do projeto. A sua estrutura de apresentação procura seguir de perto a estrutura de apresentação do capítulo anterior: aquisição, acesso programático e visualização. No entanto, ao contrário deste, cada secção ou subsecção retrata as particularidades de implementação de uma determinada funcionalidade ou requisito do projeto. Dada a dinâmica evolutiva das várias plataformas com que este projeto tinha de lidar, foram várias as alterações ao nível de arquitetura, já durante a fase de implementação. Assim, foi também necessário rever (e muitas vezes recomeçar) a implementação de determinadas funcionalidades. Alguns destes contratempos são também expressos no decorrer deste capítulo.

4.1 Obtenção do título

É necessário verificar pelo menos duas vezes por dia o conteúdo do EPG, isto porque o EPG do SAPO é atualizado duas vezes. O conteúdo será filtrado apenas para as 48 horas seguintes. Este valor é configurável no ficheiro de propriedades do projeto. Para descarregar o conteúdo do EPG é usada a API do SAPO[1]. Este serviço oferece a programação existente de cada canal especificamente pedido. Não é possível, com apenas um pedido, obter todos os programas, num determinado espaço temporal, de todos os canais. Assim, a lista com os nomes dos canais a pesquisar é preparada antecipadamente. Através da API do EPG do SAPO é obtida a lista de canais existentes no Meo. Depois de obtida essa lista, é feita uma filtragem, que consiste em eliminar os canais que não contenham filmes nem séries na sua programação. Após esta filtragem inicial, são selecionados quarenta e sete canais em cento e noventa e sete. Esta seleção foi feita manualmente, criando um ficheiro com os nomes dos canais cuja a programação se pretende descarregar. Desta forma, torna-se possível introduzir ou retirar novos canais, sem ser necessário fazer uma nova implementação no servidor. A listagem apresentada no exemplo 4.1 ilustra a resposta, por parte dos serviços de EPG do SAPO, ao pedido de informações sobre o agendamento de programação do canal Sic.

```
Título = Primeiro Jornal
Descrição = Toda a atualidade sobre as principais notícias do dia.
Start time = 2011-03-31 13:00:00
Duração = 5400
```

[1] http://services.sapo.pt/EPG/

```
Título = Alma Gémea - Ep. 62
Descrição = Serena diz que Luna é parte dela e que precisa descobrir qual é a sua
missão. Elias diz que não pode ajudá-la porque pode ser muito doloroso para ela.
Cristina assume que conhecia Guto e que errou ao não contar antes que ele e
Serena estavam se encontrando.
Start time = 2011-03-31 14:30:00
Duração = 3600

Título = Boa Tarde T2 - Ep. 14
Descrição = O entretenimento chega todas as tardes, com Conceição Lino. A boa
disposição desta apresentadora promete muita animação para começar a tarde
da melhor forma.
Start time = 2011-03-31 15:30:00
Duração = 9900

Título = Ti Ti Ti - Ep. 69
Descrição = Desirée rompe com Armandinho e devolve o colar que ganhou. Julinho
descobre que Eduardo tem uma namorada. Luisa comunica que vai passar um tem
-po fora do país. Ariclenes suspeita que Jacques lhe recomendou Gigi e diz a Chi
-co que precisa se livrar dela.
Start time = 2011-03-31 18:15:00
Duração = 3600

Título = Escrito Nas Estrelas - Ep. 109
Descrição = Vitória/Viviane conversa com Daniel, acalmando seu espírito.
Athael, Francisca e Seth se unem a Viviane e ficam satisfeitos quando Daniel se
afasta. Jardel chantageia Dalva para ficar hospedado na casa dela. Ricardo agradece
a Jane por ter feito todo o procedimento de Vitória/Viviane.
Start time = 2011-03-31 19:15:00
Duração = 2700

Título = Jornal Da Noite
Descrição = Toda a actualidade sobre as principais notícias do dia.
Start time = 2011-03-31 20:00:00
Duração = 6000
```

Exemplo 4.1: Demonstração da resposta do EPG a um pedido de um canal num determinado *timestamp*

4.2 Obtenção do URI

Inicialmente pensou-se em obter todos os filmes existentes no DBpedia. Para isso, foi criado um módulo que, através de *queries* SPARQL, obtinha os URIs de todos os filmes e séries existentes no DBpedia. Desta forma, seria armazenada toda a informação existente no DBpedia correspondente a filmes e séries. Depois de coletados todos os dados, verificava-se a existência de novos elementos desde a última pesquisa. Esta hipótese rapidamente foi abandonada dados os problemas de performance na resposta a um *query*. Além disso, cada resposta estava limitada a um máximo de 10000 resultados, obrigado a *query* a apresentar também a lógica necessária à paginação dos resultados. Atendendo aos constrangimentos de

performance e ao crescimento do DBpedia, o tempo de espera pela execução deste conjunto de operações seria muito superior ao desejado.

Como apenas serão pesquisados dados na *triplestore* que se encontram no EPG do SAPO, é relevante pesquisar apenas pelos dados que surgem no EPG. Todos estes dados são filtrados. Por exemplo, a informação do EPG congrega o nome das séries com o número da temporada e com o número do episódio associado. No entanto, esta informação é irrelevante para o cenário apresentado: apenas são recolhidos dados sobre a série em geral. Mais concretamente, no caso de "Ossos T5 - Ep. 13", o algoritmo remove a indicação da temporada e do episódio "T5 - Ep. 13" ficando apenas com o nome da série "Ossos". Posteriormente, todos os nomes repetidos são eliminados, colocando apenas nomes distintos numa pilha.

Depois de listados os nomes é necessário obter o URI correspondente ao nome desse filme/série para se poder adquirir o seu conteúdo. Para a obtenção do URI é usado o serviço de *lookup* do DBpedia. Optou-se por este serviço devido à velocidade de resposta, que em média demora um segundo, em comparação a *queries* SPARQL (no anexo A.7 podem ser verificados os tempos de resposta dos *queries* SPARQL). No entanto, ambos os módulos foram implementados, dado que o serviço de *lookup* se encontrar muitas vezes *offline*.

Como o DBpedia não se trata exclusivamente de uma base de dados de filmes e séries é necessário restringir a pesquisa. O serviço de *lookup* suporta a especificação da classe que se desejada procurar. Deste modo, melhora-se significativamente o tempo de resposta e obtém-se o que se procura com maior precisão. Para isso, no URL é inserido *QueryClass = TelevisionShow* e *QueryClass= film*, por exemplo:
lookup.dbpedia.org/api/search.asmx/KeywordSearch?QueryClass = film&queriestring = titanic. Desta forma, apenas são obtidos os filmes Titanic e é descartada a série Titanic.

Uma desvantagem do serviço *lookup* em comparação aos *queries* SPARQL é que o serviço simplesmente procura pelo nome na propriedade *foaf:name* definido na ontologia do DBpedia. O grande problema nesta pesquisa é que na propriedade *foaf:name* é armazenado o nome original da série/filme. Em sentido inverso, no EPG apenas se consegue ter acesso aos nomes traduzidos para português.

4.3 Obtenção do título original

Como nem todos os filmes/séries são uma tradução literal ao título, não é possível o uso de um dicionário para obter a sua tradução. Desta forma, torna-se necessário encontrar o título original do filme correspondente ao título fornecido. O Freebase tem uma base de dados muito mais atualizada comparativamente ao DBpedia. Uma das ferramentas que a API do Freebase contém é um serviço de pesquisa. Para usar este serviço basta ligar-se ao URL *https://www.googleapis.com/freebase/v1/search?indent=true&key="123456789"-&query="nome_do_filme"*. É devolvida uma resposta em *JavaScript Object Nation* (JSON)[2] com todos os resultados obtidos. No anexo A.1 encontra-se uma pesquisa pelo nome "A origem".

Pelas razões explicadas no capítulo anterior, na secção 4.7.2, é selecionado o primeiro filme que surge na resposta em JSON. É por fim através do serviço de *lookup* do DBpedia obtido o URI correspondente ao filme que se pretende. Este URI é introduzido na pilha de URIs a processar. Para além deste módulo, foi criado um outro, que apesar de não ser utilizado devido à sua lentidão na obtenção de resultados, conseguia que estes tivessem uma taxa de

[2]http://www.json.org

sucesso mais elevada. Este módulo é composto pelos *queries* que se encontram exemplificados em 4.2 e 4.3.

```
{
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?film WHERE {
?film rdf:type <http://dbpedia.org/ontology/Film> .
?film foaf:name \"" + title + "\"@en .

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?film WHERE {
?film rdf:type <http://dbpedia.org/ontology/TelevisionShow> .
?film foaf:name \"" + title + "\"@en .
}
```

Exemplo 4.2: Queries para obter o conteúdo através de um título em inglês

```
{
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?film WHERE {
?film rdf:type <http://dbpedia.org/ontology/Film> ."
?film rdfs:label ?movie .
"FILTER regex(?movie, \"" + title + "\"@pt, \"i\")}

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?film WHERE {
?film rdf:type <http://dbpedia.org/ontology/TelevisionShow> ."
?film rdfs:label ?movie .
FILTER regex(?movie, \"" + title + "\"@pt, \"i\")}
}
```

Exemplo 4.3: queries para obter o conteúdo através de um título em português

O sucesso obtido através destes *queries* é superior em comparação ao serviço de *lookup*. Não foi usado o *endpoint* SPARQL do DBpedia porque a maior parte das pesquisas foram feitas com títulos em português. O tempo médio de resposta dos *queries* que contêm literais em português é de cinquenta e cinco segundos, em comparação com o segundo do serviço *lookup*.

4.4 Obtenção do conteúdo

4.4.1 Verificação da ligação com a triplestore

Quando é inserido algum título de um filme ou série na pilha, é iniciado o processo de obtenção do conteúdo correspondente a um determinado filme ou série. O primeiro procedimento que este processo realiza é tentar ligar-se à *triplestore*. No caso de ainda não existir, é criada uma. Caso exista, é verificado se o *dataset* está envolvido numa transação. Se estiver, aguardam-se três segundos e verifica-se novamente, até ser possível realizar a operação em causa. Desta forma, evitam-se os problemas de concorrência. O adiamento indefinido também é prevenido dado que se garante que todas as *threads* que usam este *dataset* gerem o início e o fim da conexão com o *dataset*.

4.4.2 URIs renomeados

Por vezes são obtidos URIs que já não existem, ou seja, em vez de darem acesso ao respetivo conteúdo, apenas apresentam o novo URI, como e apresentado no anexo A.3. Este novo endereço obtido pela resposta HTTP será o novo URI correspondente ao objeto pretendido. Este novo URI já contém toda a informação associada que é pretendida.

No caso da pesquisa seja feita através de *queries* SPARQL, é devolvido um RDF com a seguinte propriedade *http://dbpedia.org/ontology/wikiPageRedirects*. Desta forma, é possível verificar que o URI pesquisado foi renomeado.

4.4.3 Obtenção da informação ligada ao respetivo URI

Depois de adquirido o URI, este é usado para aceder à informação necessária, sobre o recurso em causa, para este projeto. Como toda a sua informação é guardada localmente, o URI é alterado com um *host* local. Por exemplo, *http://dbpedia.org/resource/Unstoppable_(2010_film)* será armazenado como *http://hostname/resource/Unstoppable_(2010_film)* em que *hostname* será o nome do domínio do servidor onde se encontra a aplicação. Este novo objeto criado localmente terá uma nova propriedade *http://www.w3.org/2002/07/owl#sameAs*, sendo uma propriedade existente na ontologia do DBpedia. Com esta propriedade o objeto será ligado ao URI original de onde foram obtidos os dados. Desta forma, é sempre possível verificar de onde provieram os dados, podendo facilmente atualizar os dados correspondentes.

4.4.4 Adicionar géneros de fontes externas

Em toda a informação associada a cada filme/série obtida pelo DBpedia, nem sempre se encontra o respetivo género, e quando este se obtém, por vezes está incompleto. Deste modo, é necessário a sua obtenção através de fontes externas. Na descrição do filme/série encontram-se várias propriedades, em que uma delas é o *id* correspondente ao filme/série no *dataset* do Freebase. Numa das propriedades *#sameAs*, encontra-se um valor, por exemplo *http://rdf.freebase.com/ns/m.076zy_g*, este valor é efetivamente o *id* correspondente ao filme *Unstoppable* no Freebase. Esta informação encontra-se no DBpedia devido ao Freebase ser uma das suas fontes, mas, a verdade é que, nem toda a informação foi importada por completo, como por exemplo, o género.

A ferramenta mais importante e indispensável para lidar com o Freebase é a linguagem MQL. São criados dois *queries* MQL (exemplo 4.4), de forma a obter o nome e os géneros associados ao filme/série. Para a distinção entre filme ou série, a ontologia do DBpedia possibilita o conhecimento de que o objeto corresponde a um filme ou a uma série. Esta distinção é feita através da verificação se um dos tipos do objeto é um filme "http://dbpedia.org/ontology/Film"ou uma série "http://dbpedia.org/ontology/TelevisionShow". Sendo assim, reduz-se significativamente o tempo de pesquisa no Freebase.

Relativamente ao género, para cada filme/série, o Freebase devolve o identificador do género, bem como a sua designação. Para a obtenção do género é usada uma biblioteca para *JAVA* do Freebase, sendo possível através dela executar um *query* ilustrados no exemplo 4.4.

Como foi explicado na secção 3.1, os géneros não podem ser diretamente guardados na forma como são obtidos no Freebase. Estes dados devem ser guardados com base na taxonomia pré-definida, para isso, foi criado um novo módulo. Este módulo, apenas tem como função encontrar e modificar, caso seja necessário, os géneros obtidos do Freebase iguais aos géneros da lista pré-definida. Será um autómato que contém três estados.

O primeiro estado, verifica se existe algum género com o nome idêntico àquele que se encontra pré-definido.

O segundo estado, separa todas as palavras. Por exemplo o Freebase contém muitos géneros com o nome *"Romantic comedy"* ou *"Romance Film"*; no primeiro caso, cada palavra corresponde a um género, enquanto que no segundo caso, apenas interessará a primeira palavra. Posteriormente, são eliminados todos os nomes que se encontram repetidos. Por fim, é verificado se os novos nomes provenientes da separação se encontram na lista de géneros.

O terceiro estado reduz o género à sua forma raiz. Para a deteção de sinónimos procuraram-se várias técnicas de *stemming*. A solução encontrada foi a mais simples em comparação com todas as soluções experimentadas, tendo sido a que obteve melhores resultados. O método encontrado divide a palavra ao meio, mais um caractere. Este termo é depois comparado com o início das palavras reservadas (nome de géneros). Por exemplo, *romance* é uma das palavras reservadas e *romantic* trata-se de um género que surge no Freebase; com este método, *romantic* passa a ser *roman* que é o início de *romance*. Este método devido à sua simplicidade, transmitia bastante insegurança mas depois de inúmeros testes foi verificado que para a lista existente obtinham-se sempre resultados corretos. O único género que não tem mais de cinco caracteres é o *War*, sendo necessário criar exclusivamente um comparador para este género. Foi testado um *stemmer* denominado como *snowball*[3] devido à sua reputação. Mas para este caso não seria a ferramenta ideal porque, por exemplo, a resposta deste *stemmer* à palavra *romantic* é *romant* e à palavra *romance* é *romanc* e, assim, o problema não é resolvido. Foi também pensado usar APIs de certos sites, como por exemplo: *http://www.programmableweb.com/apitag/thesaurus* ou *http://developer.dictionary.com/products/synonyms* para obter sinónimos, o que revelou algum sucesso. Apesar de se obterem melhores resultados em comparação com os resultados do *snowball*, o primeiro método continua a ser o mais fiável.

```
o(      "id", id,
        "type", "/film/film",
        "name", null,
        "genre", a(o(
        "id", null,
        "name", null)));


o(      "id", id,
        "type", "/tv/tv_program",
        "name", null,
        "genre", a(o(
        "id", null,
        "name", null)));
```

Exemplo 4.4: *queries* MQL para obtenção do género associado

4.4.5 Adicionar a classificação

A classificação (*rating*) é um dado fundamental para distinguir entre dois filmes/séries com o mesmo nome. Como o EPG apenas fornece o nome, este dado é fundamental para ser possível prever qual o filme/série se trata.

[3] http://snowball.tartarus.org

A classificação não se encontra no DBpedia, por este motivo foi necessário obter esse dado através de uma fonte externa. Como foi referido anteriormente, o IMDB não foi considerado porque não é permitido extrair dele qualquer informação. Sendo assim, a fonte externa eleita foi o *Rotten Tomatoes*. Através da classificação encontrada pode-se deduzir se o filme é o mais popular, permitindo assim a distinção entre dois nomes iguais.

Obtendo uma chave de identificação através do registo no *Rotten Tomatoes* pode-se efetuar um pedido, sendo devolvido um JSON como ilustrado em anexo A.4. Da informação recebida é extraído apenas o *audience score*.

O *Rotten Tomatoes* tem como grande desvantagem o facto de ser unicamente vocacionado para filmes. Assim, as séries não são classificadas. Dadas as limitações temporais na execução deste trabalho, a pesquisa por uma fonte de dados para a classificação de séries deverá ser considerada como trabalho futuro.

4.5 Criação de uma ontologia

A classe género, essencial para o sistema de recomendação, não existe na ontologia do DBpedia, tendo sido necessária a sua especificação. Para isso, foi usado um editor *open source* de ontologias: o Protégé. Com este editor, é possível criar uma ontologia visualmente e exportá-la no formato OWL. Este formato é um requisito necessário para a indexação da triplestore através de uma ferramenta externa a este projeto. Também para a classificação foi necessário criar uma nova classe. O DBpedia apesar de ter anunciado que futuramente iria introduzir a classificação dos filmes e séries, até a data ainda não introduziu nenhuma classe análoga à classificação. Para facilitar a fusão da ontologia criada com a ontologia do DBpedia foi usado uma super classe idêntica a uma classe existente na ontologia do DBpedia. Vendo a figura 4.1, verifica-se que *Movie* e *Film*, são super classes, ambas equivalentes e existentes na ontologia do DBpedia *http://dbpedia.org/ontology/film e http://dbpedia.org/ontology/movie*. Para ligar estas super classes às classes criadas localmente *http://hostname/ontology/rating* e *http://hostname/ontology/genre* foram criadas as seguintes propriedades:

- `http://hostname/property/hasRating`
- `http://hostname/property/isRatingOf`
- `http://hostname/property/hasGender`
- `http://hostname/property/isGenderOf`

Desta forma, obtêm-se propriedades simétricas, o que significa que o filme/série está relacionado com a classificação ou com o género e vice-versa.

Todas as outras classes ligadas à classe *film* que não foram nomeadas e que se encontram na figura 4.1 correspondem efetivamente a classes importadas do DBpedia. A classe Género apenas aceita valores pré-determinados. Para isso, foram criados vinte e três *individuals*, desta forma a ontologia apenas permite ligar a classe género a literais pré-determinados. Os vinte e três *individuals* são os seguintes:

- ACTION
- ANIMATION
- COMEDY
- ADVENTURE
- BIOGRAPHY
- CRIME

- DOCUMENTARY
- DRAMA
- FAMILY
- FANTASY
- FILM-NOIR
- HISTORY
- HORROR
- MISTERY
- MUSICAL
- ROMANCE
- SCI-FI
- SUPERNATURAL
- SPORTS
- THRILLER
- WAR
- WESTERN
- MUSIC

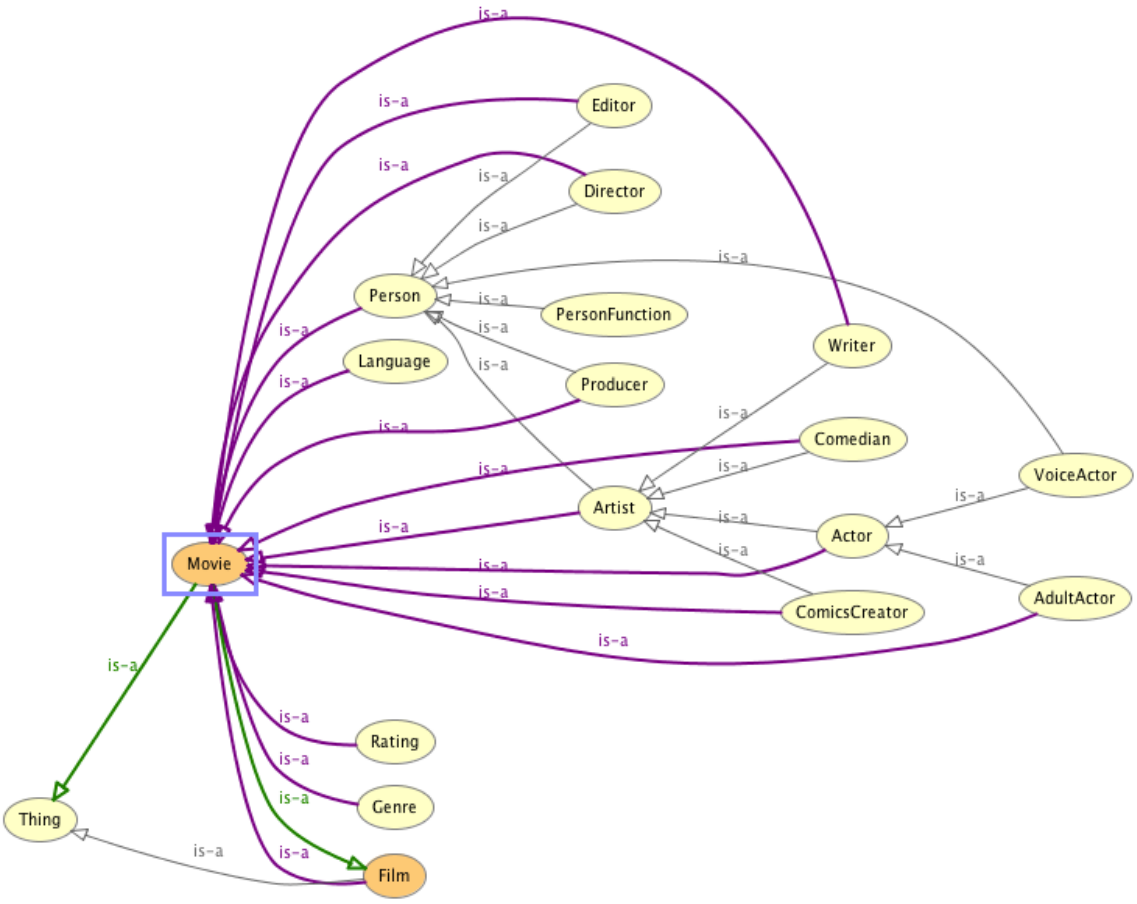

Figura 4.1: Ontologia criada no Protégé.

4.6 Sistema de threads

Este projeto pode obter um número muito grande de informação. Por esse motivo, foi necessário desenvolver um sistema que dê prioridade aos dados que se encontrem diretamente ligados a filmes/séries que estão a ser pesquisados. Somente depois é que são pesquisados os dados que estão indiretamente ligados. Assim, a informação principal encontra-se disponível mais rapidamente. Para isso, foi necessário criar uma *thread pool* tornando o projeto mais

rápido. A ideia é que cada *thread* seja especializada em determinada função. As funções (e portanto as *threads*) criadas permitem obter:

- Todas as propriedades;
- Todos os atores que participaram num dado filme;
- A informação correspondente ao segundo grau de profundidade.

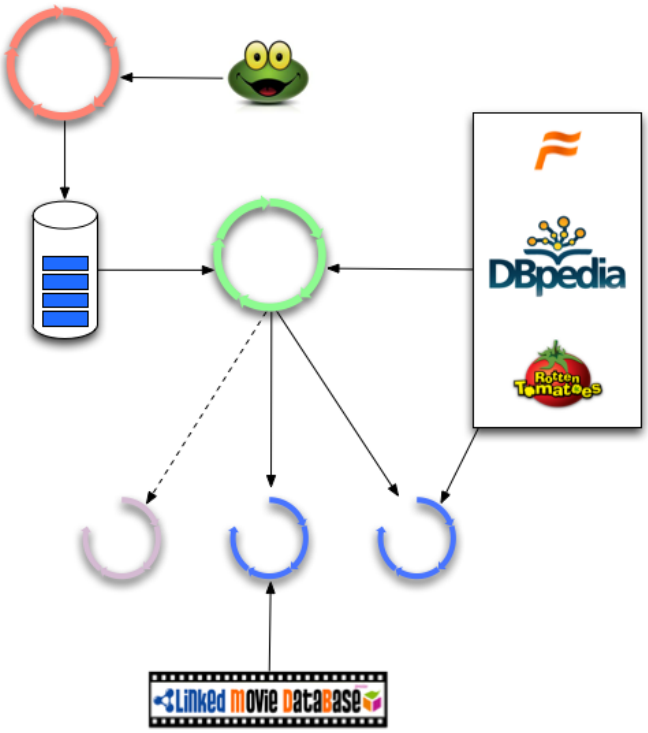

Figura 4.2: Esquema de threads

4.6.1 Thread de gestão de propriedades

Apesar de atualmente excluída da solução final, a função deste tipo de *thread* é verificar, no objeto de que se está a obter informação, se todas as propriedades associadas já se encontram na ontologia local. No caso de uma das propriedades obtidas não se encontrar guardada localmente, esta é guardada. A propriedade guardada é igual à propriedade proveniente do DBpedia, somente o *hostname* seria o correspondente ao próprio servidor onde a aplicação se encontra. Por outras palavras, com esta funcionalidade ativa, a ontologia criada seria uma cópia parcial da ontologia do DBpedia apenas com um *hostname* diferente. Esta ideia surgiu, devido à ontologia do DBpedia ser muito extensa, contendo inúmeras propriedades que não são necessárias para filmes e séries. Por outro lado e porque todos os parâmetros ontológicos associados a filmes e séries podem representar qualquer "coisa", poderá sempre surgir uma propriedade diferente que ainda não se encontre na ontologia. Na figura 4.2 esta *thread* encontra-se indicada pela circunferência a cinzento.

Atendendo a que inicialmente não se tinha pensado no fornecimento de uma API a terceiros, os tempos de acesso não eram uma preocupação. O processo de aquisição de informação seria feito sempre com tempo (pesquisas *offline*). No entanto, a possibilidade de pesquisa por parte de terceiros, de informação ainda não consolidada na triplestore local, leva a que os tempos de acesso se tornem incomportáveis. Depois de ter surgido este requisito foi necessário uma otimização para se conseguir o melhor tempo de resposta possível. Após ponderação, a solução encontrada passou pela desativação desta *thread*. A ontologia criada deixa de ser uma cópia parcial da ontologia do DBpedia e passa a ser efetivamente a ontologia do DBpedia mais a junção de uma ontologia própria que contém somente duas propriedades que não existem na ontologia do DBpedia. Esta alteração melhorou significativamente a performance da aplicação. Conceptualmente uma ontologia deve-se usar por completo e não parcialmente, mesmo que se considere que existem propriedades e objetos que não sejam usados.

4.6.2 Thread de aquisição de atores

O segundo tipo de *thread* tem a responsabilidade de obter todos os atores correspondentes ao filme/série que está a ser processado. Na figura 4.2 esta *thread* encontra-se indicada pela primeira circunferência azul, ligada à linked movie database. Como o DBpedia ainda se encontra incompleto, verifica-se que os atores que são referenciados no DBpedia sobre um filme/série são menos que no LinkedMDB. Apesar de uma das fontes principais do DBpedia ser o LinkedMDB, os dados que se encontram no DBpedia ainda não têm correspondência total com o linkedMDB. Esta migração é lenta por ser necessário efetuar um alinhamento de ontologias, dado que a ontologia do LinkedMDB não é idêntica à do DBpedia. Na troca de emails com programadores responsáveis do DBpedia, verificou-se que alguma da informação importada por eles é feita de uma forma manual, o que justifica a lentidão da migração.

Depois de obter todos os atores correspondentes a um filme/série através do DBpedia, esta *thread* é responsável por verificar se existem mais atores no LinkedMDB. Caso existam, esta *thread* cria um objeto para este novo ator, com todos os dados que lhe correspondem mas com a ontologia respetiva ao DBpedia. Desta forma, é mantida uma triplestore coerente e acessível a qualquer sistema.

4.6.3 Thread de aquisição do segundo grau de profundidade

O terceiro tipo de *thread* é o responsável por obter dados com uma profundidade de ordem dois. Esta *thread* verifica todos os objetos diretamente ligados ao objeto principal (objeto para o qual está a ser apurada toda a informação associada) e, por sua vez, extrai toda a informação associada a esses objetos. Na figura 4.2 esta *thread* encontra-se indicada pela segunda circunferência azul. Na figura 4.3 é demonstrada a hierarquia dos objetos, bem como o grau correspondente.

O desenvolvimento deste tipo de *thread* foi exigente, pois inicialmente foi pensado que a ordem de profundidade seria previamente definida pelo utilizador. Mas este requisito não foi implementado porque oferecia muitas desvantagens face ao tempo de resposta. Como existem propriedades simétricas, reflexivas e transitivas seria vital preparar o software para ter noção dos fatores de recursividade, para não entrar em *loop*.

Este tipo de *thread* apenas importa dados que correspondem ao DBpedia. Os dados não importados correspondem a ficheiros com um tamanho significativo, como por exemplo: imagens, músicas e *traillers*. Ao evitar esta importação são reduzidos os recursos necessários,

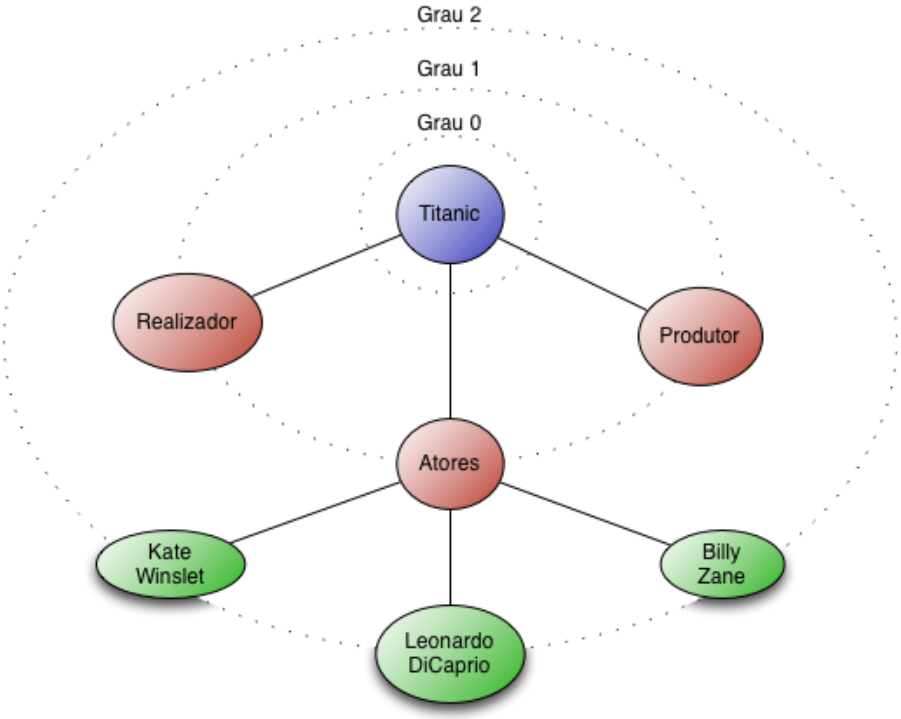

Figura 4.3: Graus de profundidade

como também é otimizado o tempo de resposta. No caso de ser necessário um destes dados não importados, através da ligação que é guardada no primeiro grau, facilmente são obtidos. Desta forma, o utilizador tem a liberdade de selecionar o que realmente deseja armazenar.

4.6.4 Thread de gestão/controlo

As *threads* são todas criadas, inicializadas e geridas por uma *thread* principal. Na figura 4.2 esta *thread* encontra-se indicada pela circunferência verde.

Esta *thread* é responsável por retirar o conteúdo da pilha onde estão guardados todos os URIs. Ela somente está ativa quando a *thread* que introduz URIs na pilha (representada na figura 4.2 pela circunferência laranja) está desativa e vice-versa. O objetivo é que estas duas *threads* nunca se encontrem a funcionar simultaneamente. Depois de obter o URI , a *thread* principal tem a responsabilidade de obter toda a informação que se encontra no primeiro grau, de alterar o *hostname* local e, por fim, de ativar as *threads* anteriormente anunciadas. Como esta *thread* é mais rápida que as restantes, para não criar *threads* de um modo perigoso, o sistema foi limitado a dez *threads* no máximo, em que duas delas estão constantemente ativas. No caso da *thread pool* estar cheia, esta *thread* espera três segundos. Esta validação e compasso de espera repete-se até ser possível continuar.

Toda a concorrência a nível da escrita de *logs* é gerida pelo biblioteca *log4j*.

4.7 Application Programming Interfaces (APIs)

Como foi referido no capítulo anterior, este projeto tem quatro APIs disponíveis. Estas APIs encontram-se sobre um servidor Apache. Para a criação das APIs usou-se *Java API for RESTful Web Services* (JAX-RS) que fornece suporte para a criação de *Web services* de acordo com a arquitetura ReST.

A função destas APIs como o seu funcionamento é descrito nas seguintes secções:

4.7.1 Visualizar e explorar

A navegação livre é realizada através do *"http://hostname/CineFetch/NavigationServlet-.html"*, sendo possível visualizar todo o conteúdo existente na *triplestore* através do *browser*. Todos os URIs apresentados podem ser acedidos para a visualização dos respetivos conteúdos. Desta forma, é possível procurar a informação desejada não através do nome numa pesquisa direta, mas sim, navegando através das relações com outras entidades na *triplestore*.

4.7.2 Pesquisar pelo nome

Para pesquisar pelo nome, basta introduzir o URL *"http://hostname/ CineFetch/tripleStore/search/title"* e serão pesquisadas na *triplestore* local todas as séries e filmes que contenham o título fornecido. A resposta enviada para o cliente é um XML com os parâmetros pré-definidos; no anexo A.5 encontra-se um exemplo de uma resposta ao pedido *Inception*.

Inicialmente é procurado o título em português na *triplestore* local e no caso de não se obter nenhum resultado, é pesquisado o título em Inglês. Continuando a ter insucessos na pesquisa, é efetuado um pedido para verificar no DBpedia a existência de algum URI correspondente ao título em questão. Se for obtido algum resultado, é guardado o conteúdo correspondente e, posteriormente, serão efetuados os processos anteriormente explicados.

Para além de se obter a informação associada ao objeto pedido, para a criação deste XML, em determinadas situações, é necessário navegar para um nível superior para obter um literal. Mais concretamente, a informação diretamente associada nem sempre se trata de literais mas sim de URIs. Por exemplo, o realizador ligado diretamente ao objeto filme é sempre um URI, para a obtenção do o seu nome é necessário navegar para o URI realizador.

4.7.3 Visualização de um objeto

Caso seja necessário visualizar o filme com mais detalhe basta introduzir o nome em Inglês obtido em XML e colocar no fim do endereço *http://hostname/CineFetch/resource/*, sendo este o URI local do objeto; desta forma é dada uma resposta em HTML de todo o conteúdo associado a este objeto.

Depois de concluído o projeto, não se irá obter um pedido de algum filme ou série que não se encontre na *triplestore*. Dado que outros projetos dependentes deste obtêm os nomes dos filmes/séries através do EPG do SAPO, tem-se a garantia que o mesmo nome já foi processado. Atendendo à validade do EPG (7 dias), os dados são obtidos com quarenta e oito horas de antecedência da hora atual, permitindo que todo o processamento ocorra com alguma antecedência da sua utilização.

4.7.4 Armazenamento

Para o armazenamento de dados basta enviar um ficheiro XML, no formato idêntico ao anexo A.5, para o URL *"http://hostname/CineFetch/tripleStore/search/send"*. Serão guardados na *triplestore* apenas os dados fornecidos.

4.8 Visualização

Foi usada uma ferramenta em *javascript* para gerar visualizações interativas[4] através do *browser*, esta ferramenta requer um ficheiro JSON para revelar a hierarquia dos nós. Para gerar este ficheiro foi necessário traduzir um ficheiro no formato RDF para JSON.

A leitura de um ficheiro RDF é feita através da *framework* JENA, este ficheiro RDF apenas contém as informações sobre um único objeto. A tradução é elaborada segundo as seguintes regras: o recurso é o nó raiz, as propriedades são as ligações e os objetos são os nós adjacentes. Ao nó raiz e às ligações são associados os nomes correspondentes. Para o caso dos nós ligados ao nó raiz, é feito o mesmo procedimento com a exceção do objeto se tratar do ator.

Como um filme contém um grande número de atores terá imensos nós ligados, tornando-se muito confuso e ilegível, como se pode verificar na figura 4.4. Esta figura é gerada automaticamente através da *framework* JENA. Relativamente à visualização, optou-se por não usar esta figura porque no caso de um objeto conter muitos dados associados, é impraticável uma boa visualização e compreensão da figura. Assim, solução encontrada para uma melhor visualização foi ligar todos os atores a um *blank node*, como explicado na secção 2.2. Desta forma, conseguem-se ter os nós mais dispersos, facilitando a sua perceção, como é ilustrado na figura 3.3.

4.9 Problemas encontrados

Na elaboração deste projeto surgiram vários problemas. A razão fulcral para o surgimento destes problemas deve-se ao facto desta tecnologia semântica ainda ser muito recente.

Um dos exemplos é a biblioteca Apache Jena (explicado na secção 2.8.5) que é usada neste projeto; esta, ainda tem algumas limitações. No caso de ambientes *multi-threaded* ainda não está otimizada, existindo ainda problemas críticos por resolver, apesar de a última versão experimentada já mostrar indícios de melhorias. Numa versão anterior (a versão que foi usada inicialmente neste projeto foi a 0.9.0-incubating) problemas de concorrência não eram evitados, sendo possível ler e escrever livremente na mesma TDB com *threads* diferentes. Inicialmente, este projeto tinha uma variável partilhada por todas as *threads* do sistema para saber se o *dataset* estaria "aberto"para escrita ou para leitura. Depois, em Agosto surgiu uma nova atualização de JENA, esta comparada às duas atualizações feitas anteriormente ofereceu mais trabalho, pelo facto de muitos dos métodos usados se encontrarem agora *deprecated*[5]. Até à versão 0.92 apenas eram criados modelos somente em memória volátil, com esta atualização é sempre necessário criar um modelo através do *dataset*. Inicialmente a aplicação criava primeiro o modelo em memória e só depois é que o guardava no *dataset*. Desta forma o *dataset* encontra-se-ia "aberto"o mínimo tempo possível, apenas se usava o *dataset* quando o

[4] http://philogb.github.com/jit/index.html

[5] *http://jena.apache.org/documentation/javadoc/tdb/com/hp/hpl/jena/tdb/TDBFactory.html*

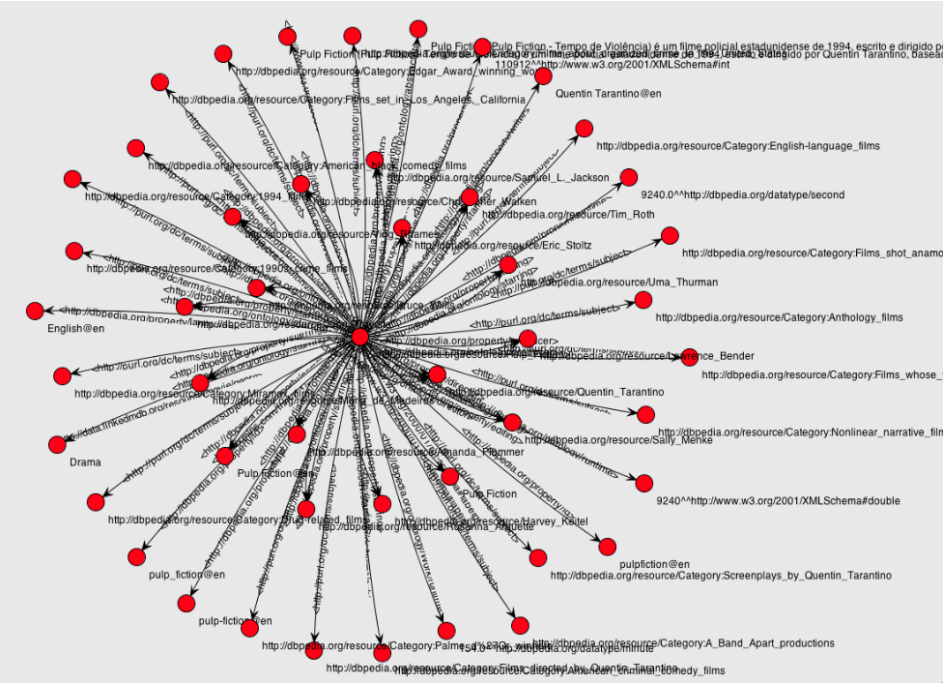

Figura 4.4: Grafo gerado através da *framework* JENA

modelo estivesse concluído. Esta alteração exigiu uma atualização em todos os métodos onde fosse necessário manusear esta informação. Mas com esta última atualização já é possível saber se o *dataset* está "aberto"para leitura ou para escrita, evitando o problema de ter uma variável partilhada por todas as *threads* existentes.

Mas o maior problema detetado existente no Apache Jena é o facto de não ser possível fazer um *rollback*. Este problema é critico, por exemplo, no momento da escrita se acontecer algum problema obrigando a escrita terminar inesperadamente irá corromper um ficheiro da *triplestore*. O grande problema é que por vezes o ficheiro corrompido é o ficheiro onde estão indexados todos os prefixos ou o ficheiro onde estão indexados os id's dos tripletos. Basta um destes dois ficheiros ser corrompido para que a *triplestore* fique automaticamente inacessível. Para evitar esta situação é necessário que, de cada vez que o *dataset* seja "aberto"para escrita, seja feito um *backup* da *triplestore*. Não basta apenas guardar os ficheiros de indexação porque a própria gravação de novos elementos podem alterar os índices da *triplestore* anterior, para isso será obrigatório fazer uma cópia integral de toda a *triplestore*.

Um outro problema encontrado refere-se ao facto da TDB não guardar os prefixos quanto é usado o *tdbloader*, *issue*[6] já reportado há algum tempo.

A maior fonte de complicações foi exatamente o DBpedia. O DBpedia já existe desde 2007, mas apesar dos cinco anos de existência, ainda existem *bugs* e alterações consideráveis. Um dos *bugs* que já existe há bastante tempo é o facto dos URIs introduzidos não serem descodificados. Isto é, ao introduzir *http://dbpedia.org/resource/Inception_%28film%29* é diferente que introduzir *http://dbpedia.org/resource/Inception_(film)*[7]. Como o serviço de *lookup* do DBpedia fornece todos os URIs em *HTML URL Encoding* é necessário fazer *decoding* de to-

[6]https://issues.apache.org/jira/browse/JENA-175

[7]`http://wiki.dbpedia.org/Internationalization/Issues/Encoding?v=5c7`

dos os URIs obtidos pelo serviço de *lookup*. Por outro lado, os URIs obtidos associados a um determinado objeto já não se encontram codificados no formato URL.

O DBpedia ainda é uma ferramenta muito recente, necessitando de constantes modificações. Estas modificações provocam que as APIs estejam constantemente *offline*. Por vezes os serviços encontram-se indisponíveis durante uma semana.

Um outro problema do DBpedia é o facto dos títulos dos filmes e das séries não se encontrarem traduzidos. Apesar de conter uma *label* a indicar que o título está em português, muitas vezes este encontra-se em inglês.

Toda a informação associada a filmes e séries ainda se encontram muito incompleta. DBpedia deveria ter associado a cada filme e série informações como a classificação, banda sonora, atores, capas, entre outros dados, como foi anunciado pelo DBpedia[20]; mas muita dessa informação ainda não se encontra disponível. Uma grande melhoria que está a ser implementada é o processo que permite a conversão da informação da Wikipédia para RDF, com a consequente disponibilização na Web.

À medida que este projeto se foi desenrolando vários dos problemas já identificados foram sendo resolvidos. Apesar das alterações implicarem, por vezes, a reescrita do código, estas permitiram otimizações ao nível de:

- Migração dos URLs conforme a especificação dos tripletos;
- Servidor já aceita nomes que contenham um *slash*;
- Valores normalizados (já não aparece tempo do filme como 11640.000000 (xsd:double))

Por fim um último problema encontrado, é o facto de alguns filmes que se encontram no Wikipédia e não no DBpedia. Por exemplo o filme "O Artista", através do serviço *lookup* do DBpedia, de todos os resultados obtidos nenhum é referente ao filme mais conhecido. Até ao mês de Agosto existia um site[8] que permitia explorar o DBpedia livremente e procurar mais facilmente o que se pretendia através do *browser*. Por vezes, o filme desejado encontrava-se realmente no DBpedia mas o nome encontrava-se de um modo diferente ou simplesmente o serviço de *lookup* não o encontrava apesar de lá estar. De acordo com o exemplo referido anteriormente, relativamente ao filme "O Artista", o nome original *"The Artist"* é encontrado através do Freebase, no entanto não é encontrado pelo serviço de *lookup*.

[8] `http://dbpedia.neofonie.de/browse/`

Capítulo 5

Conclusão

Com este projeto foi possível verificar a utilidade que a semântica nos oferece. Com a semântica a pesquisa da informação torna-se mais eficiente, conseguindo obter com maior exatidão o que é procurado. O conteúdo da informação encontrada é mais elaborado e fidedigno.

Através deste projeto foi possível obter informação televisiva de uma forma mais interativa, de modo que tanto os utilizadores como as aplicações consigam encontrar a informação desejada de uma forma mais rápida e eficiente. Com este projeto é possível criar mais uma fonte de informação semântica, ajudando assim ao crescimento da Web 3.

Contudo, verifica-se que esta tecnologia é muito recente. Ao desenvolver uma aplicação utilizando a Web Semântica facilmente surgem problemas pelo facto de muitas tecnologias usadas não se encontrarem num estado estável para a sua utilização. Mas quando estas tecnologias se encontrarem numa fase mais madura, estou convicto que estas revolucionarão a Web.

Quando as fontes deste projeto se encontrarem mais estáveis, existe um conjunto de tarefas que podem ser exploradas e elaboradas, tanto ao nível de pesquisa como ao nível de exploração de dados. Neste sistema destacam-se as seguintes oportunidades de trabalho futuro: Nível de profundidade, Ambiente gráfico interativo, Backups inteligentes e automatizados e Machine learning.

Nível de profundidade: depois da ontologia do DBpedia se encontrar concluída, já se pode criar um dicionário com um número significativo de propriedades. Deste modo será possível evitar ciclos. Para isso, é necessário o conhecimento de propriedades simétricas, reflexivas e transitivas. Somente assim é possível explorar dinamicamente com níveis de profundidade aleatórios.

Ambiente gráfico interativo: no caso de clicar num dos nós no gráfico criado por esta aplicação o nó poderá deslocar-se para o centro da imagem, mas seria interessante o sistema começar a pesquisar pelas ligações que se encontram diretamente ligadas a esse nó. Os níveis de profundidade podiam ser ajustados, sendo desta forma possível configurar o tamanho desejado do grafo.

Backups inteligentes e automatizados: como a TDB não se encontra preparada para a recuperação de ficheiros, a criação de um sistema que gere um *backup* antes de ser efetuada uma escrita é algo a considerar. Uma simples cópia total da TDB seria uma possibilidade, mas seria muito lenta e requeria sempre que o *backup* fosse efetuado num servidor com o dobro do espaço de armazenamento. O ideal seria criar um sistema que apenas guardasse um

histórico das alterações feitas

e apenas guardasse o que foi alterado antes de ser efetivamente alterado.

Desta forma, no caso de ocorrer algum erro será possível a sua recuperação. Caso todas as transações sejam bem efetuadas, este histórico pode ser apagado. Assim, o sistema será rápido e não necessitará de tantos recursos.

Machine learning: para criar uma ontologia adequada à pesquisa que é feita ao próprio cliente. Apesar de ambicioso, criar um motor de pesquisa semântico que "aprenda"os nossos gostos seria um enorme avanço desta tecnologia. Pelo facto desta tecnologia ter uma ontologia associada, o sistema espera encontrar todos os objetos que ela define, mas no caso de surgir um objeto desconhecido para a ontologia, o sistema não sabe lidar com isso. Com o decorrer do tempo, uma ontologia poderá tornar-se incompleta. O ideal seria retirar a ontologia estática e criar uma dinâmica através de métodos de *machine learning*. Com *machine learning* será possível coletar informação suficiente da Web, permitindo a análise de forma a que o sistema consiga "aprender"o que existe no mundo bem como todas as relações existentes.

Bibliografia

[1] T. Segaran, C. Evans, and J. Taylor, *Programming the semantic web.* O'Reilly Media, Incorporated, 2009.

[2] J. Hebeler, M. Fisher, R. Blace, A. Perez-Lopez, and M. Dean, *Semantic Web Programming.* Wiley, 2009.

[3] S. Powers, *Practical rdf.* O'Reilly Media, Incorporated, 2003.

[4] S. Staab and R. Studer, *Handbook on ontologies.* Springer, 2009.

[5] A. Maedche and S. Staab, "Ontology learning for the semantic web," *Intelligent Systems, IEEE*, vol. 16, no. 2, pp. 72–79, 2001.

[6] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools," *University of Manchester*, no. 1.0, 2004.

[7] C. Wende, U. Abmann, and A. Bartho", *Reasoning Web.* Springer, 2010.

[8] M. Hausenblas, W. Halb, and Y. Raimond, "Building linked data for both humans and machines," in *WWW 2008 Workshop: Linked Data on the Web (LDOW2008), Beijing, China*, 2008.

[9] A. Maedche and S. Staab", "Ontology learning," *ICWSM Conference*, 2009.

[10] I. Dickinson, "Elo perdido," *Apache Jena*, 2011.

[11] R. Poli, M. Healy, and A. Kameas, *Theory and Applications of Ontology.* Springer, 2010.

[12] B. Smith, L. Goldberg, A. Ruttenberg, and M. Glick, "Ontology and the future of dental research informatics," *J Am Dent Assoc*, vol. 141, no. 10, pp. 1173–1175, 2010.

[13] A. Owens, "Clustered triple storage for jena," *HP labs Bristol*, 2008.

[14] N. Noy, "Ontology mapping and alignment," *Stanford University*, 2011.

[15] SPARQL. (2012, Maio) Sparql. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[16] L. Data. Linked data. [Online]. Available: http://linkeddata.org

[17] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "Dbpedia-a crystallization point for the web of data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 3, pp. 154–165, 2009.

[18] D. Team. (2008) Dbpedia team. [Online]. Available: http://wiki.dbpedia.org/Team

[19] DBpedia. Dbpedia 3.8 released. [Online]. Available: http://blog.dbpedia.org/2012/08/06/dbpedia-38-released-including-enlarged-ontology-and-additional-localized-versions/

[20] D. Team. (2008) Dbpedia dataset. [Online]. Available: http://wiki.dbpedia.org/Datasets#h18-3

[21] D. M. (2010) Dbpedia mappings. [Online]. Available: http://mappings.dbpedia.org/index.php/Main_Page

[22] RdfLib. (2012, Março) Rdflib. [Online]. Available: http://www.rdflib.net

[23] RdfAlchemy. (2012, Março) Rdfalchemy. [Online]. Available: http://www.openvest.com/trac/wiki/RDFAlchemy

[24] FuXi. (2012, Março) Fuxi. [Online]. Available: http://code.google.com/p/fuxi

[25] jena. Reasoners and rule engines: Jena inference supportlinked data tools. [Online]. Available: http://jena.apache.org/documentation/inference/index.html

[26] L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, *The Semantic Web - Iswc 2011.* Springer, 2011.

[27] W. Halb, Y. Raimond, and M. Hausenblas, "Comparison of triple stores."

[28] L. Sarmento, "Codebits, verbetes da sapo," 2011.

[29] BBC. (2012, Agosto) Bbc. [Online]. Available: http://www.bbc.co.uk/blogs/bbcinternet/2012/04/sports_dynamic_semantic.html

[30] M. Bastian and S. Heymann, "Gephi: An open source software for exploring and manipulating networks," *FZI Research Center for Information Technologies*, 2009.

[31] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[32] A. Singhal. (2012, Agosto) Introducing the knowledge graph: things, not strings. [Online]. Available: http://googleblog.blogspot.pt/2012/05/introducing-knowledge-graph-things-not.html

[33] "Oobian, product description and implementation methodology," Tech. Rep., 2009.

Anexos A

Anexos

A.1 Resposta da API Frebase


```
{
  "status": "200 OK",
  "result": [
    {
      "mid": "/m/0661ql3",
      "id": "/wikipedia/it_id/2539984",
      "name": "Inception",
      "notable": {
        "name": "Film",
        "id": "/film/film"
      },
      "lang": "en",
      "score": 441.390808
    },
    {
      "mid": "/m/03qcfvw",
      "id": "/en/g_i_joe_rise_of_cobra",
      "name": "G.I. Joe: The Rise of Cobra",
      "notable": {
        "name": "Film",
        "id": "/film/film"
      },
      "lang": "en",
      "score": 143.888687
    },
    {
      "mid": "/m/0btyf5z",
      "id": "/wikipedia/en_title/Rise_of_the_Planet_of_the_Apes",
      "name": "Rise of the Planet of the Apes",
      "notable": {
        "name": "Film",
        "id": "/film/film"
      },
      "lang": "en",
      "score": 140.594330
    },
    {
      "mid": "/m/01322j_",
```

```
    "name": "A Origem Do Mal",
    "notable": {
      "name": "Musical Recording",
      "id": "/music/track"
    },
    "lang": "en",
    "score": 93.053253
  },
  {
    "mid": "/m/0syj04",
    "name": "A Origem Da Felicidade",
    "notable": {
      "name": "Musical Recording",
      "id": "/music/track"
    },
    "lang": "en",
    "score": 72.094627
  },
  {
    "mid": "/m/010934d",
    "name": "A Origem do Drama",
    "notable": {
      "name": "Musical Recording",
      "id": "/music/track"
    },
    "lang": "en",
    "score": 65.479202
  },
  {
    "mid": "/m/0lhvqdq",
    "name": "A Origem Do Mal",
    "notable": {
      "name": "Release track",
      "id": "/music/release_track"
    },
    "lang": "en",
    "score": 60.660229
  },
  {
    "mid": "/m/0kwhjc_",
    "name": "A Origem do Drama",
    "notable": {
      "name": "Release track",
      "id": "/music/release_track"
    },
    "lang": "en",
    "score": 58.725018
  },
  {
    "mid": "/m/0lb7d2f",
    "name": "A Origem Da Felicidade",
    "notable": {
      "name": "Release track",
      "id": "/music/release_track"
    },
    "lang": "en",
    "score": 58.725018
```

```
},
{
  "mid": "/m/03cwf_t",
  "id": "/en/denominacao_de_origem_controlada",
  "name": "Denomina\u00e7\u00e3o de Origem Controlada",
  "lang": "en",
  "score": 10.632284
},
{
  "mid": "/m/0v5_h3",
  "name": "Cavaleiros do Fogo da Origem (ou A Com\u00e9dia dos Erros)",
  "notable": {
    "name": "Musical Recording",
    "id": "/music/track"
  },
  "lang": "en",
  "score": 7.102197
},
{
  "mid": "/m/044z2s4",
  "id": "/wikipedia/images/commons_id/359869",
  "name": "Fotografia do assalto a banco, em 1973, que deu origem ao nome
      da s\u00edndrome",
  "notable": {
    "name": "Content",
    "id": "/type/content"
  },
  "lang": "en",
  "score": 5.886347
},
{
  "mid": "/m/0lpg21f",
  "name": "Cavaleiros do Fogo da Origem (ou A Com\u00e9dia dos Erros)",
  "lang": "en",
  "score": 5.533295
},
{
  "mid": "/m/01khjf",
  "id": "/en/appellation",
  "name": "Appellation",
  "lang": "en",
  "score": 4.525324
},
{
  "mid": "/m/06kl9xr",
  "id": "/en/vozes_da_origem",
  "name": "Vozes da origem",
  "notable": {
    "name": "Book",
    "id": "/book/book"
  },
  "lang": "en",
  "score": 4.481700
},
{
  "mid": "/m/0682st7",
  "id": "/en/origem_das_especies",
```

```
      "name": "Origem das esp\u00e9cies",
      "notable": {
        "name": "Book",
        "id": "/book/book"
      },
      "lang": "en",
      "score": 4.481700
    },
    {
      "mid": "/m/04s4sr",
      "id": "/en/origin",
      "name": "Origin",
      "lang": "en",
      "score": 3.328227
    },
    {
      "mid": "/m/01m1k",
      "id": "/en/common_descent",
      "name": "Common descent",
      "lang": "en",
      "score": 3.088909
    },
    {
      "mid": "/m/0gmq11",
      "id": "/en/dinosaur-bird_connection",
      "name": "Origin of birds",
      "lang": "en",
      "score": 2.893749
    },
    {
      "mid": "/m/02x95z",
      "id": "/en/origin_of_replication",
      "name": "Origin of replication",
      "lang": "en",
      "score": 2.866088
    }
  ],
  "cursor": 20,
  "cost": 14,
  "hits": 95
}
```


Exemplo A.1: Resposta à pesquisa do nome *A Origem* através do Freebase

A.2 Resposta do Freebase


```
{
  "code":           "/api/status/ok",
  "result": [{
    "genre": [
      {
        "id":   "/en/thriller",
        "name": "Thriller"
      },
      {
        "id":   "/m/02kdv5l",
        "name": "Action"
```

```
      },
      {
        "id":    "/m/03btsm8",
        "name": "Action/Adventure"
      },
      {
        "id":    "/en/drama_film",
        "name": "Drama"
      }
    ],
    "id":    "/m/076zy_g",
    "name": "Unstoppable",
    "type": "/film/film"
  }],
  "status":          "200 OK",
  "transaction_id": "cache;cache03.p01.sjc1:8101;2012-09-18T01:38:03Z;0007"
}
```


Exemplo A.2: Resposta à pesquisa dos géneros *Unstoppable* atravez do Freebase

A.3 Redirect

```
http://dbpedia.org/resource/Inception_(film)
```

```
About to connect() to dbpedia.org port 80 (#0)
*   Trying 194.109.129.58...
* connected
* Connected to dbpedia.org (194.109.129.58) port 80 (#0)
> GET /resource/Inception_(film) HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL
    /0.9.8r zlib/1.2.5
> Host: dbpedia.org
> Accept: */*
>
< HTTP/1.1 303 See Other
< Date: Sat, 22 Sep 2012 22:28:48 GMT
< Content-Type: text/html; charset=UTF-8
< Connection: keep-alive
< Server: Virtuoso/06.04.3132 (Linux) x86_64-generic-linux-glibc25-64  VDB
< Accept-Ranges: bytes
< Location: http://dbpedia.org/page/Inception_(film)
< Content-Length: 0
<
* Connection #0 to host dbpedia.org left intact
* Closing connection #0
```

Exemplo A.3: Redirect

A.4 Resposta do Rotten Tomatoes


```
{
  "total": 2,
  "movies": [{
    "id": "770672122",
    "title": "Toy Story 3",
```

```
"year": 2010,
"mpaa_rating": "G",
"runtime": 103,
"critics_consensus": "Deftly blending comedy, adventure, and honest emotion
    , Toy Story 3 is a rare second sequel that really works.",
"release_dates": {
  "theater": "2010-06-18",
  "dvd": "2010-11-02"
},
"ratings": {
  "critics_rating": "Certified Fresh",
  "critics_score": 99,
  "audience_rating": "Upright",
  "audience_score": 91
},
"synopsis": "Pixar returns to their first success with Toy Story 3. The
    movie begins with Andy leaving for college and donating his beloved toys
     -- including Woody (Tom Hanks) and Buzz (Tim Allen) -- to a daycare.
    While the crew meets new friends, including Ken (Michael Keaton), they
    soon grow to hate their new surroundings and plan an escape. The film
    was directed by Lee Unkrich from a script co-authored by Little Miss
    Sunshine scribe Michael Arndt. ~ Perry Seibert, Rovi",
"posters": {
  "thumbnail": "http://content6.flixster.com/movie/11/13/43/11134356_mob.
      jpg",
  "profile": "http://content6.flixster.com/movie/11/13/43/11134356_pro.jpg
      ",
  "detailed": "http://content6.flixster.com/movie/11/13/43/11134356_det.jpg
      ",
  "original": "http://content6.flixster.com/movie/11/13/43/11134356_ori.jpg
      "
},
"abridged_cast": [
  {
    "name": "Tom Hanks",
    "characters": ["Woody"]
  },
  {
    "name": "Tim Allen",
    "characters": ["Buzz Lightyear"]
  },
  {
    "name": "Joan Cusack",
    "characters": ["Jessie the Cowgirl"]
  },
  {
    "name": "Don Rickles",
    "characters": ["Mr. Potato Head"]
  },
  {
    "name": "Wallace Shawn",
    "characters": ["Rex"]
  }
],
"alternate_ids": {"imdb": "0435761"},
"links": {
  "self": "http://api.rottentomatoes.com/api/public/v1.0/movies/770672122.
```

```
            json",
        "alternate": "http://www.rottentomatoes.com/m/toy_story_3/",
        "cast": "http://api.rottentomatoes.com/api/public/v1.0/movies/770672122/
            cast.json",
        "clips": "http://api.rottentomatoes.com/api/public/v1.0/movies/770672122/
            clips.json",
        "reviews": "http://api.rottentomatoes.com/api/public/v1.0/movies
            /770672122/reviews.json",
        "similar": "http://api.rottentomatoes.com/api/public/v1.0/movies
            /770672122/similar.json"
      }
    }],
    "links": {
      "self": "http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Toy+
          Story+3&page_limit=1",
      "next": "http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Toy+
          Story+3&page_limit=1&page=2"
    },
    "link_template": "http://api.rottentomatoes.com/api/public/v1.0/movies.json?q
        ={search-term}&page_limit={results-per-page}&page={page-number}"
}
```


Exemplo A.4: Resposta à pesquisa do nome *Toy Story* atravez do Rotten Tomatoes

A.5 Tempos de queries SPARQL

Query	Tempo (ms)
?film ?p ?s	780
?film rdf:type <http://dbpedia.org/ontology/Film>; ?p ?s	802
?film rdf:type <http://dbpedia.org/ontology/Film>; foaf:name "Titanic"@en; ?p ?s	674
?film ?p ?s; rdf:type <http://dbpedia.org/ontology/Film>; foaf:name "Titanic"@en	7217
?film rdf:type <http://dbpedia.org/ontology/Film>; ?p ?s; foaf:name "Titanic"@en	1763
?film rdf:type <http://dbpedia.org/ontology/Film>; foaf:name "Titanic"@en; ?p ?s	1263

Tabela A.1: Demonstração da diferença temporal com as ordem do Query em várias formas

A.6 Resposta da API

<TVDATABASE>

```
    <NamePT>A Origem</NamePT>
    <NameEN>Inception</NameEN>
    <Format>Movie</Format>
    <Gender>Romance</Gender>
    <Time>148.0</Time>
    <Director>Christopher</Director>
    <starring>Inception é um filme estadunidense de çãfico ícientfica çlanado em
        2010. Escrito, dirigido e produzido pelo âbritnico Christopher Nolan,
        estrela o longa Leonardo DiCaprio, Ken Watanabe, Joseph Gordon-Levitt,
        Marion Cotillard, Ellen Page, Tom Hardy, Cillian Murphy, Dileep Rao, Tom
        Berenger e Michael Caine. DiCaprio faz o papel de Dom Cobb, um ãladro
        especializado em extrair çõinformaes do inconsciente dos seus alvos
        durante o sonho. Incapaz de visitar seus filhos, Cobb tem a chance de êv-
        los em troca de um último trabalho: fazer a çãinsero, plantar a origem de
        uma ideia na mente de um rival de seu cliente. O desenvolvimento do
        filme çcomeou em 2001, quando Nolan escreveu um tratamento de 80 ápginas
        sobre õladres de sonhos, apresentando a ideia para a Warner Bros. A
        óhistria foi originalmente concebida como um filme de çãao, inspirado nos
        conceitos de sonhos úlcidos e çãincubao de sonhos. Sentindo que
        precisava de mais êexperincia em filmes de grande çoramento, Nolan optou
        por outros longas e decidiu trabalhar mais seis meses no roteiro de
        Inception antes que a Warner o comprasse, em fevereiro de 2009. As
        filmagens passaram por seis ípases e quatro continentes, çcomeando em
        óTquio, em junho de 2009, e terminando no áCanad, em novembro do mesmo
        ano. O compositor Hans Zimmer ôcomps a trilha sonora do filme usando
        trechos da çãcano "Non, je ne regrette rien", de Edith Piaf. Inception
        teve um çoramento oficial de US$ 160 õmilhes, dividido o custo entre a
        Warner e a Legendary Pictures. A çãreputao e o sucesso de Nolan obtidos
        com The Dark Knight ajudou a conseguir mais US$ 100 õmilhes para
        çãdivulgao. Inception teve sua éestria em Londres, no dia 8 de julho de
        2010, e foi çlanado em cinemas convencionais e IMAX no dia 14 de julho de
        2010. Impulsionado pela boa ícrtica, arrecadou US$ 823.576.195 na
        bilheteria mundial. Foi indicado a oito Oscars, incluindo Melhor Filme,
        vencendo em quatro categorias, Melhor Fotografia, Melhores Efeitos
        Visuais, Melhor çãEdio de Som e Melhor Mixagem de Som.</starring>
    <rating>86</rating>
</TVDATABASE>
```

Exemplo A.5: XMLAPI

A.7 Média de tempo de pesquisa SPARQL

Idioma	Tempo demorado (segundos)
Inglês	2
Portugês	55

Tabela A.2: Média de tempo de pesquisa SPARQL